Computers and Fluids 120 (2015) 173–186

Contents lists available at ScienceDirect

Computers and Fluids

journal homepage: www.elsevier.com/locate/compfluid

Heterogeneous computing on mixed unstructured grids with PyFR

F.D. Witherden*, B.C. Vermeire, P.E. Vincent

Department of Aeronautics, Imperial College London, SW7 2AZ United Kingdom

ARTICLE INFO

Article history:
Received 3 October 2014
Revised 17 July 2015
Accepted 21 July 2015
Available online 31 July 2015

Keywords:
High methods
Flux reconstruction
Heterogeneous computing
Cylinder flow

ABSTRACT

PyFR is an open-source high-order accurate computational fluid dynamics solver for unstructured grids. In this paper we detail how PyFR has been extended to run on mixed element meshes, and a range of hardware platforms, including heterogeneous multi-node systems. Performance of our implementation is benchmarked using pure hexahedral and mixed prismatic-tetrahedral meshes of the void space around a circular cylinder. Specifically, for each mesh performance is assessed at various orders of accuracy on three different hardware platforms; an NVIDIA Tesla K40c GPU, an Intel Xeon E5-2697 v2 CPU, and an AMD FirePro W9100 GPU. Performance is then assessed on a heterogeneous multi-node system constructed from a mix of the aforementioned hardware. Results demonstrate that PyFR achieves performance portability across various hardware platforms. In particular, the ability of PyFR to target individual platforms with their 'native' language leads to significantly enhanced performance *cf.* targeting each platform with OpenCL alone. PyFR is also found to be performant on the heterogeneous multi-node system, achieving a significant fraction of the available FLOP/s. Finally, each mesh is used to undertake nominally fifth-order accurate long-time simulations of unsteady flow over a circular cylinder at a Reynolds number of 3900 using a cluster of NVIDIA K20c GPUs. Long-time dynamics of the wake are studied in detail, and results are found to be in excellent agreement with previous experimental/numerical data. All results were obtained with PyFR v0.2.2, which is freely available under a 3-Clause New Style BSD license (see www.pyfr.org).

1. Introduction

There is an increasing desire amongst industrial practitioners of computational fluid dynamics (CFD) to perform scale-resolving simulations of unsteady compressible flows within the vicinity of complex geometries. However, current generation industry-standard CFD software—predominantly based on first- or second-order accurate Reynolds Averaged Navier–Stokes (RANS) technology—is not well suited to the task. Henceforth, over the past decade there has been significant interest in the application of high-order methods for mixed unstructured grids to such problems. Popular examples of high-order schemes for mixed unstructured grids include the discontinuous Galerkin (DG) method, first introduced by Reed and Hill [1], and the spectral difference (SD) methods originally proposed under the moniker 'staggered-gird Chebyshev multidomain methods' by Kopriva and Kolias in 1996 [2] and later popularised by Sun et al. [3]. In 2007 Huynh [4] proposed the flux reconstruction (FR) approach; a unifying framework for high-order schemes on unstructured grids that incorporates both the nodal DG schemes of [5] and, at least for a linear flux function, any SD scheme. 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. This locality makes such schemes extremely good candidates for acceleration using either the vector units of modern CPUs or graphics processing units (GPUs) [6–8]. There exist a variety of approaches for writing accelerated codes. These include directive based methodologies such as OpenMP 4.0 and OpenACC, and language frameworks such as OpenCL and CUDA. We contend, however, that at the time of writing no single approach is *performance portable* across all major hardware platforms, and that codes desiring cross-platform portability must therefore incorporate support for multiple approaches. Further, there is also a growing interest from the scientific community in *heterogeneous computing* whereby multiple platforms are employed simultaneously to solve a problem. The promise of heterogeneous computing is improved resource utilisation on systems with a mix of hardware. Such systems are becoming increasingly common.

PyFR is a high-order FR code for solving the Euler and compressible Navier–Stokes equations on mixed unstructured grids [8]. Written in the Python programming language PyFR incorporates backends for C/OpenMP, CUDA, and OpenCL. It is therefore capable of running on conventional CPUs, as well as GPUs from both NVIDIA and AMD, as well as heterogeneous mixtures of the aforementioned. The objective of this paper is to demonstrate the ability of PyFR to

* Corresponding author. Tel.: +44 02075945108.
E-mail address: freddie.witherden08@imperial.ac.uk, freddie@witherden.org (F.D. Witherden).

http://dx.doi.org/10.1016/j.compfluid.2015.07.016

perform high-order accurate unsteady simulations of flow on mixed unstructured meshes using a heterogeneous hardware platform—demonstrating the concept of 'heterogeneous computing from a homogeneous codebase'. Specifically, performance of our implementation is benchmarked using pure hexahedral and mixed prismatic-tetrahedral meshes of the void space around a circular cylinder on three different hardware platforms; an NVIDIA Tesla K40c GPU, an Intel Xeon E5-2697 v2 CPU, and an AMD FirePro W9100 GPU. Performance is then assessed on a heterogeneous multi-node system constructed from a mix of the aforementioned hardware. Finally, each mesh is used to undertake nominally fifth-order accurate long-time simulations of unsteady flow over a circular cylinder at a Reynolds number of 3900, and accuracy is assessed.

The paper is structured as follows. The flux reconstruction approach is presented in Section 2 with a unified description of DG correction functions being given in Section 3. In Section 4 we provide an overview of the PyFR codebase. Details of the cylinder test case are given in Section 5. Single node performance is assessed in Section 6 while heterogeneous multi-node performance is considered in Section 7. The accuracy of PyFR for the cylinder test case is evaluated in Section 8. Finally, in Section 9 conclusions are drawn.

2. Flux reconstruction

For a detailed overview of the multidimensional flux reconstruction approach the reader is referred to Witherden et al. [8] and the references therein. In this section we shall describe the approach within the context of the three dimensional Navier–Stokes equations. Throughout this section a convention in which dummy indices on the right hand side of an expression are summed. For example $C_{ijk} = A_{ijl}B_{ilk} \equiv \sum_l A_{ijl}B_{ilk}$ where the limits are implied from the surrounding context. All indices are assumed to be zero-based. In conservative form the three dimensional Navier–Stokes equations can be expressed as

$$\frac{\partial u_\alpha}{\partial t} + \nabla \cdot \mathbf{f}_\alpha = 0, \tag{1}$$

where α is the *field variable* index, $u = u(\mathbf{x}, t) = \left\{\rho, \rho v_x, \rho v_y, \rho v_z, E\right\}$ is the solution, ρ is the mass density of the fluid, $\mathbf{v} = (v_x, v_y, v_z)^T$ is the fluid velocity vector, and E is the total energy per unit volume. For a perfect gas the pressure, p, and total energy can be related by the ideal gas law

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

where $\gamma = c_p/c_v$. The flux can be written as $\mathbf{f}_\alpha = \mathbf{f}_\alpha(u, \nabla u) = \mathbf{f}^{(\text{inv})} - \mathbf{f}^{(\text{vis})}$ where the inviscid terms are given by

$$\mathbf{f}^{(\text{inv})} = \begin{Bmatrix} \rho v_x & \rho v_y & \rho v_z \\ \rho v_x^2 + p & \rho v_y v_x & \rho v_z v_x \\ \rho v_x v_y & \rho v_y^2 + p & \rho v_z v_y \\ \rho v_x v_z & \rho v_y v_z & \rho v_z^2 + p \\ v_x(E + p) & v_y(E + p) & v_z(E + p) \end{Bmatrix}, \tag{3}$$

and the viscous terms are given according to

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

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

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

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.

To solve this system using flux reconstruction we start by rewriting Eq. (1) as a first order system

$$\frac{\partial u_\alpha}{\partial t} + \nabla \cdot \mathbf{f}_\alpha(u, \mathbf{q}) = 0, \tag{7a}$$

$$\mathbf{q}_\alpha - \nabla u_\alpha = 0, \tag{7b}$$

where $\mathbf{q}$ is an auxiliary variable.

Take $\mathcal{E}$ to be the set of supported element types in $\mathbb{R}^3$; for PyFR these are hexahedra, prisms and tetrahedra. Consider using these various elements types to construct a conformal mesh of a domain $\mathbf{\Omega}$. Inside of each element, $\mathbf{\Omega}_{\text{en}}$, we require that

$$\frac{\partial u_{en\alpha}}{\partial t} + \nabla \cdot \mathbf{f}_{en\alpha} = 0, \tag{8a}$$

$$\mathbf{q}_{en\alpha} - \nabla u_{en\alpha} = 0. \tag{8b}$$

It is convenient, for reasons of both mathematical simplicity and computational efficiency, to work in a transformed space. We accomplish this by introducing, for each element type, a standard element $\hat{\mathbf{\Omega}}_e$ which exists in a transformed space, $\tilde{\mathbf{x}} = (\tilde{x}_0, \tilde{x}_1, \tilde{x}_2)^T$. Next, we assume the existence of a mapping function for each element such that

$$x_i = \mathcal{M}_{eni}(\tilde{\mathbf{x}}), \qquad \mathbf{x} = \boldsymbol{\mathcal{M}}_{en}(\tilde{\mathbf{x}}),$$
$$\tilde{x}_i = \mathcal{M}^{-1}_{eni}(\mathbf{x}), \qquad \tilde{\mathbf{x}} = \boldsymbol{\mathcal{M}}^{-1}_{en}(\mathbf{x}),$$

along with the relevant Jacobian matrices

$$\mathbf{J}_{en} = J_{enij} = \frac{\partial \mathcal{M}_{eni}}{\partial \tilde{x}_j}, \quad J_{en} = \det\mathbf{J}_{en},$$
$$\mathbf{J}^{-1}_{en} = J^{-1}_{enij} = \frac{\partial \mathcal{M}^{-1}_{eni}}{\partial x_j}, \quad J^{-1}_{en} = \det\mathbf{J}^{-1}_{en} = \frac{1}{J_{en}}.$$

These definitions provide us with a means of transforming quantities to and from standard element space. Taking the transformed solution, flux, and gradients inside each element to be

$$\tilde{u}_{en\alpha} = \tilde{u}_{en\alpha}(\tilde{\mathbf{x}}, t) = J_{en}(\tilde{\mathbf{x}})u_{en\alpha}(\boldsymbol{\mathcal{M}}_{en}(\tilde{\mathbf{x}}), t), \tag{9a}$$

$$\tilde{\mathbf{f}}_{en\alpha} = \tilde{\mathbf{f}}_{en\alpha}(\tilde{\mathbf{x}}, t) = J_{en}(\tilde{\mathbf{x}})\mathbf{J}^{-1}_{en}(\boldsymbol{\mathcal{M}}_{en}(\tilde{\mathbf{x}}))\mathbf{f}_{en\alpha}(\boldsymbol{\mathcal{M}}_{en}(\tilde{\mathbf{x}}), t), \tag{9b}$$

$$\tilde{\mathbf{q}}_{en\alpha} = \tilde{\mathbf{q}}_{en\alpha}(\tilde{\mathbf{x}}, t) = \mathbf{J}^T_{en}(\tilde{\mathbf{x}})\mathbf{q}_{en\alpha}(\boldsymbol{\mathcal{M}}_{en}(\tilde{\mathbf{x}}), t), \tag{9c}$$

and letting $\tilde{\nabla} = \partial/\partial\tilde{x}_i$, it can be readily verified that

$$\frac{\partial u_{en\alpha}}{\partial t} + J^{-1}_{en}\tilde{\nabla} \cdot \tilde{\mathbf{f}}_{en\alpha} = 0, \tag{10a}$$

$$\tilde{\mathbf{q}}_{en\alpha} - \tilde{\nabla}u_{en\alpha} = 0, \tag{10b}$$

as required.

We next proceed to associate a set of solution points with each standard element. For each type $e \in \mathcal{E}$ take $\{\tilde{\mathbf{x}}^{(u)}_{e\rho}\}$ to be the chosen set of points where $0 \le \rho < N^{(u)}_e(\wp)$, and $\wp$ is the polynomial order. These points can then be used to construct a nodal basis set $\{\ell^{(u)}_{e\rho}(\tilde{\mathbf{x}})\}$ with the property that $\ell^{(u)}_{e\rho}(\tilde{\mathbf{x}}^{(u)}_{e\sigma}) = \delta_{\rho\sigma}$. To obtain such a set we first take $\{\psi_{e\sigma}(\tilde{\mathbf{x}})\}$ to be an orthonormal basis which spans a selected order $\wp$ polynomial space defined inside $\hat{\mathbf{\Omega}}_e$. Next we compute the elements of the generalised Vandermonde matrix $\mathcal{V}_{e\rho\sigma} = \psi_{e\rho}(\tilde{\mathbf{x}}^{(u)}_{e\sigma})$. With these a nodal basis set can be constructed as $\ell^{(u)}_{e\rho}(\tilde{\mathbf{x}}) = \mathcal{V}^{-1}_{e\rho\sigma}\psi_{e\sigma}(\tilde{\mathbf{x}})$. Along with the solution points inside of each element we also define a set of flux points on $\partial\hat{\mathbf{\Omega}}_e$. We denote the flux

points for a particular element type as $\{\tilde{\mathbf{x}}^{(f)}_{e\rho}\}$ where $0 \leq \rho < N^{(f)}_e(\wp)$. Let the set of corresponding normalised outward-pointing normal vectors be given by $\{\hat{\tilde{\mathbf{n}}}^{(f)}_{e\rho}\}$. It is critical that each flux point pair along an interface share the same coordinates in physical space. For a pair of flux points eρn and e$'\rho'$n$'$ at a non-periodic interface this can be formalised as $\mathcal{M}_{en}(\tilde{\mathbf{x}}^{(f)}_{e\rho}) = \mathcal{M}_{e'n'}(\tilde{\mathbf{x}}^{(f)}_{e'\rho'})$.

The first step in the FR approach is to go from the discontinuous solution at the solution points to the discontinuous solution at the flux points

$$u^{(f)}_{e\sigma n\alpha} = u^{(u)}_{e\rho n\alpha}\ell^{(u)}_{e\rho}(\tilde{\mathbf{x}}^{(f)}_{e\sigma}), \tag{11}$$

where $u^{(u)}_{e\rho n\alpha}$ is an approximate solution of field variable α inside of the nth element of type e at solution point $\tilde{\mathbf{x}}^{(u)}_{e\rho}$. This can then be used to compute a *common solution*

$$\mathfrak{C}_\alpha u^{(f)}_{e\rho n\alpha} = \mathfrak{C}_\alpha u^{(f)}_{\widetilde{e\rho}n\alpha} = (\tfrac{1}{2} - \beta)u^{(f)}_{e\rho n\alpha} + (\tfrac{1}{2} + \beta)u^{(f)}_{\widetilde{e\rho}n\alpha}, \tag{12}$$

where β controls the degree of up/downwinding. Here we have taken $\widetilde{e\rho}n$ to be the element type, flux point number and element number of the adjoining point at the interface. Since grids are unstructured the relationship between eρn and $\widetilde{e\rho}n$ is indirect. This necessitates the use of a lookup table.

Further, associated with each flux point is a vector correction function $\mathbf{g}^{(f)}_{e\rho}(\tilde{\mathbf{x}})$ constrained such that

$$\hat{\tilde{\mathbf{n}}}^{(f)}_{e\sigma} \cdot \mathbf{g}^{(f)}_{e\rho}(\tilde{\mathbf{x}}^{(f)}_{e\sigma}) = \delta_{\rho\sigma}, \tag{13}$$

with a divergence that sits in the same polynomial space as the solution. Using these fields we can express the solution to Eq. (10b) as

$$\tilde{\mathbf{q}}^{(u)}_{e\sigma n\alpha} = \left[\hat{\tilde{\mathbf{n}}}^{(f)}_{e\rho} \cdot \tilde{\nabla} \cdot \mathbf{g}^{(f)}_{e\rho}(\tilde{\mathbf{x}})\left\{\mathfrak{C}_\alpha u^{(f)}_{e\rho n\alpha} - u^{(f)}_{e\rho n\alpha}\right\} + u^{(u)}_{ev n\alpha}\tilde{\nabla}\ell^{(u)}_{ev}(\tilde{\mathbf{x}})\right]_{\tilde{\mathbf{x}}=\tilde{\mathbf{x}}^{(u)}_{e\sigma}}, \tag{14}$$

where the term inside the curly brackets is the 'jump' at the interface and the final term is an order $\wp - 1$ approximation of the gradient obtained by differentiating the discontinuous solution polynomial. We can now compute physical gradients as

$$\mathbf{q}^{(u)}_{e\sigma n\alpha} = \mathbf{J}^{-T\,(u)}_{e\sigma n}\tilde{\mathbf{q}}^{(u)}_{e\sigma n\alpha}, \tag{15}$$

$$\mathbf{q}^{(f)}_{e\sigma n\alpha} = \ell^{(u)}_{e\rho}(\tilde{\mathbf{x}}^{(f)}_{e\sigma})\mathbf{q}^{(u)}_{e\rho n\alpha}, \tag{16}$$

where $\mathbf{J}^{-T\,(u)}_{e\sigma n} = \mathbf{J}^{-T}_{en}(\tilde{\mathbf{x}}^{(u)}_{e\sigma})$. Having solved the auxiliary equation we are now able to evaluate the transformed flux

$$\tilde{\mathbf{f}}^{(u)}_{e\rho n\alpha} = J^{(u)}_{e\rho n}\mathbf{J}^{-1\,(u)}_{e\rho n}\mathbf{f}_\alpha(u^{(u)}_{e\rho n}, \mathbf{q}^{(u)}_{e\rho n}), \tag{17}$$

where $J^{(u)}_{e\rho n} = \det\mathbf{J}_{en}(\tilde{\mathbf{x}}^{(u)}_{e\rho})$. This can be seen to be a collocation projection of the flux. With this it is possible to compute the normal transformed flux at each of the flux points

$$\tilde{f}^{(f_\perp)}_{e\sigma n\alpha} = \ell^{(u)}_{e\rho}(\tilde{\mathbf{x}}^{(f)}_{e\sigma})\hat{\tilde{\mathbf{n}}}^{(f)}_{e\sigma} \cdot \tilde{\mathbf{f}}^{(u)}_{e\rho n\alpha}. \tag{18}$$

Considering the physical normals at the flux points we see that $\mathbf{n}^{(f)}_{e\sigma n} = \mathbf{J}^{-T\,(f)}_{e\sigma n}\hat{\tilde{\mathbf{n}}}^{(f)}_{e\sigma}$ where $\mathbf{J}^{-T\,(f)}_{e\sigma n} = \mathbf{J}^{-T}_{en}(\tilde{\mathbf{x}}^{(f)}_{e\sigma})$, which is the outward facing normal vector in physical space. The normal vector can also be expressed as $n^{(f)}_{e\sigma n}\hat{\mathbf{n}}^{(f)}_{e\sigma n}$ where $n^{(f)}_{e\sigma n}$ is the magnitude. As the interfaces between two elements conform we must have $\hat{\mathbf{n}}^{(f)}_{e\sigma n} = -\hat{\mathbf{n}}^{(f)}_{\widetilde{e\sigma}n}$. With these definitions we are now in a position to specify an expression for the *common normal flux* at a flux point pair as

$$\mathfrak{F}_\alpha f^{(f_\perp)}_{e\sigma n\alpha} = -\mathfrak{F}_\alpha f^{(f_\perp)}_{\widetilde{e\sigma}n\alpha} = \mathfrak{F}^{(\text{inv})}_\alpha - \mathfrak{F}^{(\text{vis})}_\alpha, \tag{19}$$

where $\mathfrak{F}^{(\text{inv})}_\alpha$ is obtained by performing a Riemann solve on the inviscid portion of the flux and

$$\mathfrak{F}^{(\text{vis})}_\alpha = \hat{\mathbf{n}}^{(f)}_{e\sigma n} \cdot \left\{(\tfrac{1}{2} + \beta)\mathbf{f}^{(\text{vis})}_{e\sigma n\alpha} + (\tfrac{1}{2} - \beta)\mathbf{f}^{(\text{vis})}_{\widetilde{e\sigma}n\alpha}\right\} + \tau(u^{(f)}_{e\sigma n\alpha} - u^{(f)}_{\widetilde{e\sigma}n\alpha}), \tag{20}$$

with τ being a penalty parameter. The common normal fluxes in Eq. (19) can now be taken into transformed space via

$$\mathfrak{F}_\alpha\tilde{f}^{(f_\perp)}_{e\sigma n\alpha} = J^{(f)}_{e\sigma n}n^{(f)}_{e\sigma n}\mathfrak{F}_\alpha f^{(f_\perp)}_{e\sigma n\alpha}, \tag{21}$$

$$\mathfrak{F}_\alpha\tilde{f}^{(f_\perp)}_{\widetilde{e\sigma}n\alpha} = J^{(f)}_{\widetilde{e\sigma}n}n^{(f)}_{\widetilde{e\sigma}n}\mathfrak{F}_\alpha f^{(f_\perp)}_{\widetilde{e\sigma}n\alpha}, \tag{22}$$

where $J^{(f)}_{e\sigma n} = \det\mathbf{J}_{en}(\tilde{\mathbf{x}}^{(f)}_{e\sigma})$.

It is now possible to compute an approximation for the divergence of the *continuous flux*. The procedure is directly analogous to the one used to calculate the transformed gradient in Eq. (14)

$$(\tilde{\nabla} \cdot \tilde{\mathbf{f}})^{(u)}_{e\rho n\alpha} = \left[\tilde{\nabla} \cdot \mathbf{g}^{(f)}_{e\sigma}(\tilde{\mathbf{x}})\left\{\mathfrak{F}_\alpha\tilde{f}^{(f_\perp)}_{e\sigma n\alpha} - \tilde{f}^{(f_\perp)}_{\widetilde{e\sigma}n\alpha}\right\} + \tilde{\mathbf{f}}^{(u)}_{ev n\alpha} \cdot \tilde{\nabla}\ell^{(u)}_{ev}(\tilde{\mathbf{x}})\right]_{\tilde{\mathbf{x}}=\tilde{\mathbf{x}}^{(u)}_{e\rho}}, \tag{23}$$

which can then be used to obtain a semi-discretised form of the governing system

$$\frac{\partial u^{(u)}_{e\rho n\alpha}}{\partial t} = -J^{-1\,(u)}_{e\rho n}(\tilde{\nabla} \cdot \tilde{\mathbf{f}})^{(u)}_{e\rho n\alpha}, \tag{24}$$

where $J^{-1\,(u)}_{e\rho n} = \det\mathbf{J}^{-1}_{en}(\tilde{\mathbf{x}}^{(u)}_{e\rho}) = 1/J^{(u)}_{e\rho n}$.

This semi-discretised form is simply a system of ordinary differential equations in t and can be solved using one of a number of schemes.

3. Correction functions

The nature of a given FR scheme is determined by the form of the associated vector correction function [4, 9]. Here we present our methodology for obtaining the correction function corresponding to a nodal DG scheme inside of an arbitrary domain. When considering the correction function associated with a flux point, $\mathbf{g}^{(f)}_{e\rho}(\tilde{\mathbf{x}})$, it is often more convenient to use a face-local numbering scheme in which $\rho \leftrightarrow$(ij) where i denotes the face number and j the local index on this face. Let $\{\tilde{\mathbf{\Gamma}}_{ei}\}$ refer to faces of the reference element $\hat{\mathbf{\Omega}}_e$. With these the divergences of the DG correction functions can be expressed as [10, 11]

$$\nabla \cdot \mathbf{g}^{(f)}_{e(ij)}(\tilde{\mathbf{x}}) = \psi_{ek}(\tilde{\mathbf{x}})\int_{\tilde{\mathbf{\Gamma}}_{ei}} \hat{\tilde{\mathbf{n}}} \cdot \mathbf{g}^{(f)}_{e(ij)}(\tilde{\mathbf{s}})\psi_{ek}(\tilde{\mathbf{s}})\,\mathrm{d}\tilde{\mathbf{s}} \tag{25}$$

$$= \psi_{ek}(\tilde{\mathbf{x}})\int_{\tilde{\mathbf{\Gamma}}_{ei}} \ell_{eij}(\tilde{\mathbf{s}})\psi_{ek}(\tilde{\mathbf{s}})\,\mathrm{d}\tilde{\mathbf{s}}, \tag{26}$$

where $\hat{\tilde{\mathbf{n}}}$ is the outward pointing unit normal vector, and $\ell_{eij}(\tilde{\mathbf{s}})$ is the nodal basis function associated with point j on face i of the reference element e. In the second step we have utilised the fact that Eq. (13) fixes $\hat{\tilde{\mathbf{n}}} \cdot \mathbf{g}^{(f)}_{e(ij)}(\tilde{\mathbf{s}})$ at each of the flux points on the face permitting it to be substituted for an equivalent nodal basis function. Heretofore this formulation has only been employed for simplex elements—triangles and tetrahedra—however it is valid for any element type.

4. PyFR

4.1. Overview

Key functionality of PyFR v0.2.2 is summarised in Table 1. We note that PyFR achieves platform portability via use of an innovative 'backend' infrastructure.

The majority of operations within an FR time-step can be cast as matrix multiplications of the form

$$\mathbf{C} \leftarrow c_1\mathbf{A}\mathbf{B} + c_2\mathbf{C}, \tag{27}$$

Table 1
Key functionality of PyFR v0.2.2.

Dimensions	2D, 3D
Elements	Triangles, quadrilaterals, hexahedra, tetrahedra, prisms
Spatial orders	Arbitrary
Time steppers	Euler, RK4, RK45
Precisions	Single, double
Platforms	CPUs, NVIDIA GPUs, AMD GPUs
Inter-node communication	MPI
Governing systems	Euler, compressible Navier–Stokes

Table 2
Dimensions of the volume-to-surface interpolation operator matrix at orders $\wp = 1, 2, 3, 4$ for tetrahedral, prismatic, and hexahedral element types.

	Matrix dimensions			
Type	$\wp = 1$	$\wp = 2$	$\wp = 3$	$\wp = 4$
Tet	4×12	10×24	20×40	35×60
Pri	6×18	18×39	40×68	75×105
Hex	8×24	27×54	64×96	125×150

where $c_{1,2} \in \mathbb{R}$ are scalar constants, $\mathbf{A}$ is a constant operator matrix, and $\mathbf{B}$ and $\mathbf{C}$ are row-major state matrices. Within the taxonomy proposed by Goto et al. [12] the multiplications fall into the block-by-panel (GEBP) category. The specific dimensions of the operator matrices are a function of both the polynomial order $\wp$ used to represent the solution in each element of the domain, and the element type. A breakdown of these dimensions for the volume-to-surface interpolation operator matrix associated with Eq. (11) can be found in Table 2. In PyFR platform portability of the matrix multiply operations is achieved by deferring to the GEMM family of subroutines provided by a Basic Linear Algebra Subprograms (BLAS) library for the target platform.

All other operations involved in an FR time-step are point-wise, concerning themselves solely with data at an individual solution/flux point. In PyFR platform portability of these point-wise operations is achieved via use of a bespoke domain specific language based on the Mako templating engine [13]. Mako specifications of point-wise operations are converted into backend-specific low-level code for the target platform at runtime, which is then compiled, linked and loaded into PyFR.

4.2. C/OpenMP backend

The C/OpenMP backend can be used to target CPUs from a range of vendors. The BLAS implementation employed by PyFR for the C/OpenMP backend must be specified by the user at runtime. Both single- and multi-threaded libraries are supported. When a single-threaded library is specified PyFR will perform its own parallelisation. Given a state matrix $\mathbf{B}$ of dimension (K, N) the decomposition algorithm splits $\mathbf{B}$ into N_t slices of dimension (K, N/N_t) where N_t is the number of OpenMP threads. Each thread then multiplies its slice through by $\mathbf{A}$ to yield the corresponding slice of $\mathbf{C}$. A visualisation of this approach is shown in Fig. 1. For the block-by-panel multiplications encountered in FR this strategy has been found to consistently outperform those employed by multi-threaded BLAS libraries.

4.3. CUDA backend

The CUDA backend can be used to target NVIDIA GPUs of compute capability 2.0 or later. PyCUDA [14] is used to invoke the CUDA API from Python. Matrix-multiplications are performed using the cuBLAS library which ships as part of the CUDA distribution. The cuBLAS library is exclusively column-major. Nevertheless it is possible to directly multiply two row-major matrices together by utilising the fact that

$$\mathbf{C} = \mathbf{AB} \Rightarrow \mathbf{C}^T = (\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T, \tag{28}$$

and observing the effect of passing a row-major matrix to a column-major subroutine is to implicitly transpose it.

4.4. OpenCL backend

The OpenCL backend can be used to target CPUs and GPUs from a range of vendors. The PyOpenCL package [14] is used to interface OpenCL with Python. OpenCL natively supports runtime code generation. BLAS support is provided via the open source clBLAS library, which is primarily developed and supported by AMD. For GPU devices clBLAS utilises auto-tuning in order to effectively target a wide range of architectures. Performance is heavily dependent on the underlying OpenCL implementation.

4.5. Distributed memory systems

To scale out across multiple nodes PyFR utilises the Message Passing Interface (MPI). By construction the data exchanged between MPI ranks is independent of the backend being used. A natural consequence of this is that different MPI ranks can transparently utilise different backends—hence enabling PyFR to run on heterogenous multi-node systems.

5. Cylinder test case

5.1. Overview

Flow over a circular cylinder has been the focus of various experimental and numerical studies [15, 16, 21, 22, 23, 24, 25]. Characteristics of the flow are known to be highly dependent on the Reynolds number Re, defined as

$$Re = \frac{u_\infty D}{\nu}, \tag{29}$$

where u_∞ is the free-stream fluid speed, D is the cylinder diameter, and ν is the fluid kinematic viscosity. Roshko [17] identified a stable range between $Re = 40$ and 150 that is characterised by the shedding of regular laminar vortices, as well as a transitional range between $Re = 150$ and 300, and a turbulent range beyond $Re = 300$. These results were subsequently confirmed by Bloor [18], who identified a similar set of regimes. Later, Williamson [19] identified two modes of transition from two dimensional to three dimensional flow. The first, known as Mode-A instability, occurs at Re ≈ 190 and the second, known as Mode-B instability, occurs at Re ≈ 260. The turbulent

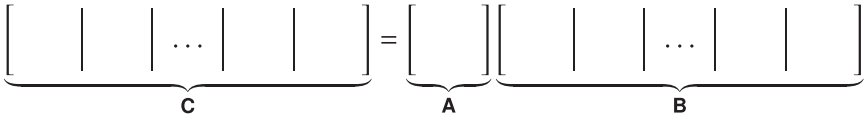
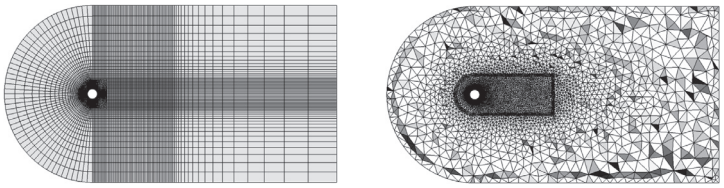
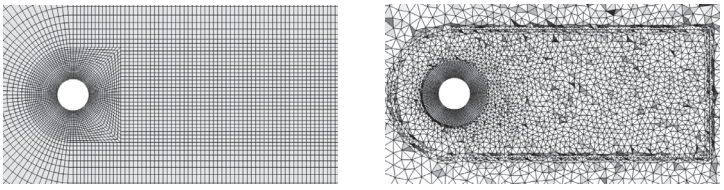


Fig. 1. How matrix multiplications are parallelised in the C/OpenMP backend of PyFR.



(a) Hexahedral, far-field. **(b)** Prism/tetrahedral, far-field.



(c) Hexahedral, wake. **(d)** Prism/tetrahedral, wake.

Fig. 2. Cutaways through the two meshes.

range beyond $Re = 300$ can be further sub-classified into the shear-layer transition, critical, and supercritical regimes as discussed in the review by Williamson [20].

In the present study we consider flow over a circular cylinder at $Re = 3900$, and an effectively incompressible Mach number of 0.2. This case sits in the shear-layer transition regime identified by Williamson [20], and contains several complex flow features, including separated shear layers, turbulent transition, and a fully turbulent wake. This test case has been the focus of a number of previous studies, both experimental and numerical [21, 22, 23, 24, 25]. Recently, Lehmkuhl et al. [26] demonstrated that the wake profile for this test case can be classified as one of two modes, a low-energy mode (Mode-L) and a high-energy mode (Mode-H). Specifically, via analysis of a very long period simulation (over 2000 convective times), they showed that the wake fluctuates between these two modes.

5.2. Domain

In the present study we use a computational domain with dimensions $[-9D, 25D]$; $[-9D, 9D]$; and $[0, \pi\mathrm{D}]$ in the stream-, cross-, and span-wise directions, respectively. The cylinder is centred at (0, 0, 0). The span-wise extent was chosen based on the results of Norberg [21], who found no significant influence on statistical data when the span-wise dimension was doubled from πD to 2πD. Indeed, a span of πD has been used in the majority of previous numerical studies [22, 23, 24], including the recent DNS study of Lehmkuhl et al. [26]. The stream-wise and cross-wise dimensions are comparable to the experimental and numerical values used by Parnaudeau et al. [25], whose results will be directly compared with ours. The overall domain dimensions are also comparable to those used for DNS studies by Lehmkuhl et al. [26]. The domain is periodic in the span-wise direction, with a no-slip isothermal wall boundary condition applied at the surface of the cylinder, and Riemann invariant boundary conditions applied at the far-field.

5.3. Mesh

The domain was meshed in two ways. The first mesh consisted of entirely structured hexahedral elements, whilst the second was unstructured, consisting of prismatic elements in the near wall boundary layer region, and tetrahedral elements in the wake and far-field. Both meshes employed quadratically curved elements, and were designed to fully resolve the near wall boundary layer region when $\wp = 4$. Specifically, the maximum skin friction coefficient was estimated *a priori* as $C_f \approx 0.075$ based on the LES results of Breuer [23].

Table 3
Approximate memory requirements of PyFR for the two cylinder meshes.

	Device memory / GiB			
Mesh	$\wp = 1$	$\wp = 2$	$\wp = 3$	$\wp = 4$
Hex	0.8	2.1	4.1	7.3
Pri/tet	1.1	2.6	4.7	7.7

The height of the first element was then specified such that when $\wp = 4$ the first solution point from the wall sits at $y+ \approx 1$, where non-dimensional wall units are calculated in the usual fashion as $y+ = u_\tau y/\nu$ with $u_\tau = \sqrt{C_f/2}u_\infty$.

The hexahedral mesh had 104 elements in the circumferential direction, and 16 elements in the span-wise direction, which when $\wp = 4$ achieves span-wise resolution comparable to that used in previous studies; as discussed by Breuer [23] and the references contained therein. The prism/tetrahedral mesh has 116 elements in the circumferential direction, and 20 elements in the span-wise direction, these numbers being chosen to help reduce face aspect ratios at the edges of the prismatic layer; which facilitates transition to the fully unstructured tetrahedral elements in the far-field. In total the hexahedral mesh contained 119, 776 elements, and the prism/tetrahedral mesh contained 79, 344 prismatic elements and 227, 298 tetrahedral elements. Both meshes are shown in Fig. 2.

5.4. Methodology

The compressible Navier–Stokes equations, with constant viscosity, were solved on each of the two meshes shown in Fig. 2. A DG scheme was used for the spatial discretisation, a Rusanov Riemann solver was used to calculate the inviscid fluxes at element interfaces, and the explicit RK45[2R+] scheme of Carpenter and Kennedy [27] was used to advance the solution in time. No sub-grid model was employed, hence the approach should be considered ILES/DNS [28, 29], as opposed to classical LES. The approximate memory requirements of PyFR for these simulations with different $\wp$ are detailed in Table 3. The total required floating point operations per RK45[2R+] time-step with different $\wp$are detailed in Fig. 3. When running with $\wp = 1$ both meshes require $\sim 1.5 \times 10^{10}$ floating point operations per time-step. This number can be seen to increase rapidly with $\wp$

6. Single-node performance

6.1. Overview

In this section we will analyse performance of PyFR on an Intel Xeon E5-2697 v2 CPU, an NVIDIA Tesla K40c GPU, and an AMD FirePro W9100 GPU. These are the individual platforms used to construct the multi-node heterogeneous system employed in Section 7.

6.2. Hardware specifications

Various attributes of the E5-2697, K40c, and W9100 are detailed in Table 4. The theoretical peaks for double precision arithmetic and memory bandwidth were obtained from vendor specifications. However, in practice it is usually only possible to obtain these theoretical peak values using specially crafted code sequences. Such sequences are almost always platform specific and seldom perform useful computations. Consequently, we also calculate and tabulate *reference* peaks. Reference peaks for double precision arithmetic are defined here as the maximum number of giga floating point operations per second (GFLOP/s) obtained while multiplying two large double precision row-major matrices using DGEMM from an appropriate BLAS library. Reference peaks for memory bandwidth are defined here as the rate, in gigabytes per second (GB/s), that data can be copied between two one gigabyte buffers. Reference peaks for the E5-2697 were obtained using DGEMM from the Intel Math Kernel Libraries (MKL) version 11.1.2, and with the E5-2697 paired with four DDR3-1600 DIMMs (with Turbo Boost enabled). Reference peaks for the K40c were obtained using DGEMM from cuBLAS as shipped with CUDA 6, with GPU Boost disabled and ECC enabled. Reference peaks

Table 4
Baseline attributes of the three hardware platforms. For the NVIDIA Tesla K40c GPU Boost was left disabled and ECC was enabled. The Intel Xeon E5-2697 v2 was paired with four DDR3-1600 DIMMs with Turbo Boost enabled.

	Platform		
	K40c	W9100	E5-2697
Arithmetic / GFLOP/s			
Theoretical peak	1430	2620	280
Reference peak	1192	890	231
Memory bandwidth / GB/s			
Theoretical peak	288	320	51.2
Reference peak	190	261	37.1
Thermal design power / W	235	275	130
Memory / GB	12	16	
Clock / MHz	745	930	3000
Transistors / Billion	7.1	6.2	4.3

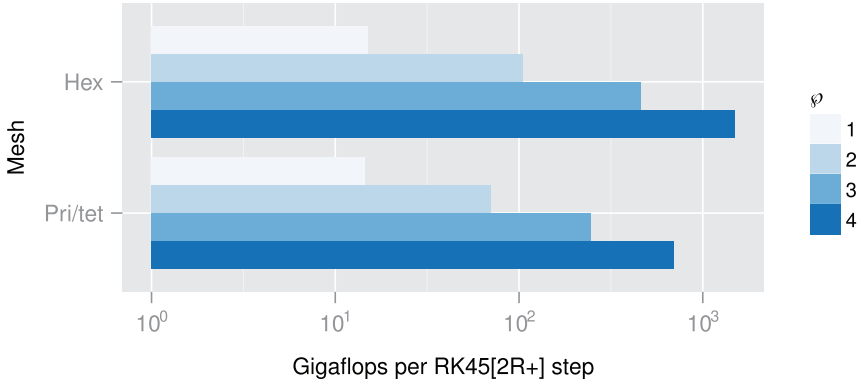


Fig. 3. Computational effort required for the 119, 776 element hexahedral mesh and the mixed mesh with 79, 344 prismatic elements and 227, 298 tetrahedral elements.

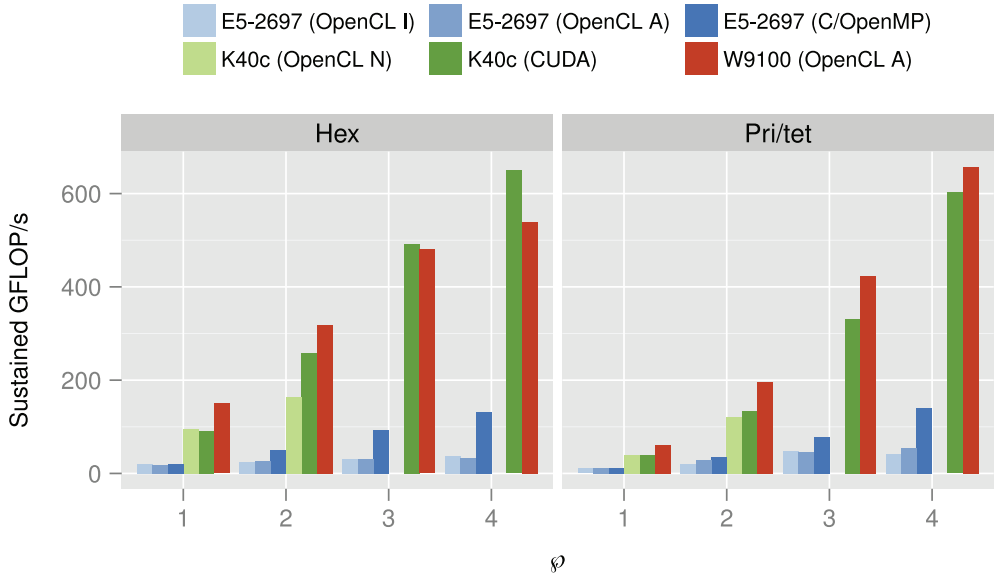


Fig. 4. Sustained performance of PyFR in GFLOP/s for the various pieces of hardware. The backend used by PyFR is given in parentheses. For the OpenCL backend the initial of the vendor is suffixed. As the NVIDIA OpenCL platform is limited to 4 GiB of memory no results are available for $\wp = 3, 4$.

Table 5
Time to evaluate $\nabla \cdot \mathbf{f}$ normalised by the total number of DOFs in the simulation which for the three dimensional Navier–Stokes equation is defined as five times the total number of solution points.

Mesh	Platform	Time per DOF / 10^{-9} s			
		$p=1$	$p=2$	$p=3$	$p=4$
Hex	E5-2697 (OpenCL I)	32.31	53.04	75.96	106.61
	E5-2697 (OpenCL A)	35.48	49.75	79.40	119.76
	E5-2697 (C/OpenMP)	32.14	28.87	27.74	31.95
	K40c (OpenCL N)	6.51	7.92		
	K40c (CUDA)	6.93	5.05	4.88	6.17
	W9100 (OpenCL A)	4.17	4.08	5.00	7.43
Pri/tet	E5-2697 (OpenCL I)	46.09	60.28	53.07	104.03
	E5-2697 (OpenCL A)	40.37	41.41	53.88	78.17
	E5-2697 (C/OpenMP)	46.32	40.68	36.74	35.53
	K40c (OpenCL N)	12.82	11.15		
	K40c (CUDA)	12.94	10.40	8.61	8.18
	W9100 (OpenCL A)	8.72	7.35	7.11	7.52

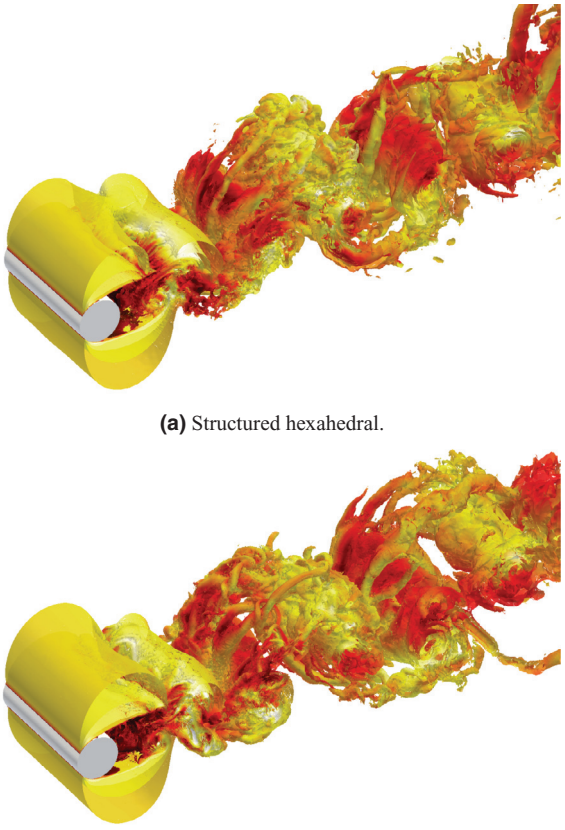
Table 6
Partition weights for the multi-node heterogeneous simulation.

Mesh	E5-2697: W9100: K40c			
	$p=1$	$p=2$	$p=3$	$p=4$
Hex	3: 27: 23	3: 27: 24	4: 24: 26	4: 24: 28
Pri/tet	5: 33: 17	5: 33: 17	5: 30: 20	5: 27: 23

for the W9100 were obtained using DGEMM from clBLAS v2.0 with version 1411.4 of the AMD APP OpenCL runtime.

We note that on the K40c ECC is implemented in software, and when enabled error-correction data is stored in global memory. A consequence of this is that when ECC is enabled there is a reduction in available memory and memory bandwidth. This partially accounts for the discrepancy observed between the theoretical and reference memory bandwidths for the K40c. We also note that for the K40c and the E5-2697, reference peaks for double precision arithmetic are in excess of 80% of their theoretical values. However, for the W9100 the reference peak for double precision arithmetic is only 34% of its theoretical value. This value is not significantly improved via the auto-tuning utility that ships with clBLAS. It is hoped that this figure will improve with future releases of clBLAS.

Finally, as an aside, we note that the number of 'cores' available on each platform have been deliberately omitted from Table 4.



(a) Structured hexahedral.

(b) Unstructured prism/tetrahedral.

Fig. 6. Instantaneous surfaces of iso-density coloured by velocity magnitude.

It is our contention that the term is both ill-defined and routinely subject to abuse in the literature. For example, the E5-2697 is presented by Intel as having 12 cores, whereas the K40c is described by NVIDIA as having 2880 'CUDA cores'. However, whereas the cores in the E5-2697 can be considered linearly independent those in the K40c can not. The rough equivalent of a CPU core in NVIDIA parlance is a 'streaming multiprocessor', or SMX, of which the K40c has 15. Additionally, the E5-2697 has support for two-way simultaneous multithreading—referred to by Intel as Hyper-Threading – permitting two threads to execute on each core. At any one instant it is therefore

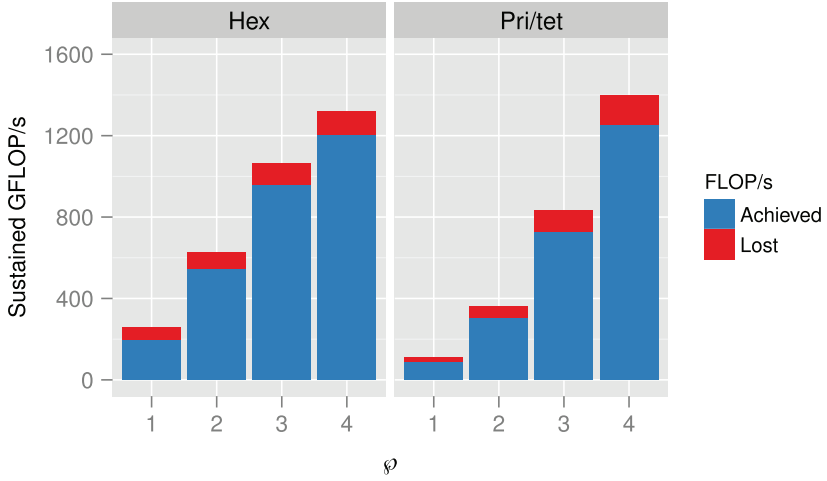


Fig. 5. Sustained performance of PyFR on the multi-node heterogeneous system for each mesh with $p = 1, 2, 3, 4$. Lost FLOP/s represent the difference between the achieved FLOP/s and the sum of the E5-2697 (C/OpenMP), K40c (CUDA), and W9100 (OpenCL A) bars in Fig. 4.

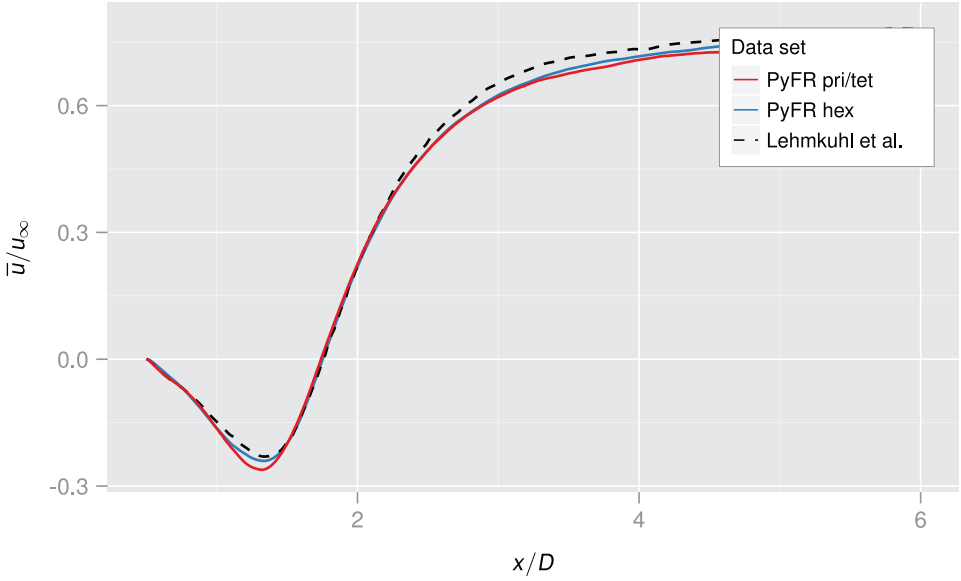


Fig. 7. Averaged wake profiles for Mode-H compared with the numerical results of Lehmkuhl et al. [26].

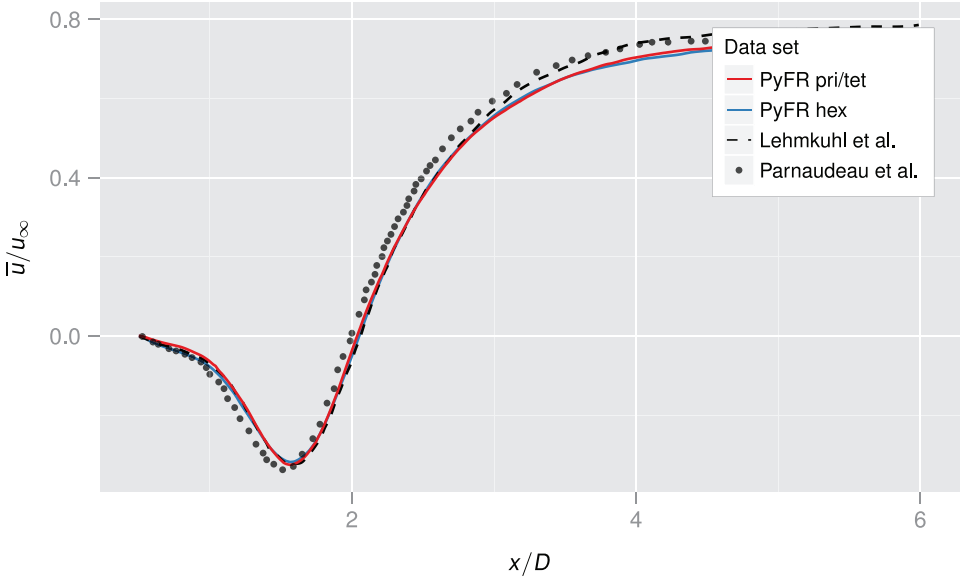


Fig. 8. Averaged wake profiles for Mode-L compared with the numerical results of Lehmkuhl et al. [26] and experimental results of Parnaudeau et al. [25].

possible to have up to 24 independent threads resident on a single E5-2697. The AMD equivalent of a CUDA core is a 'stream processor' of which the W9100 has 2816. This is not to be confused with the aforementioned streaming multiprocessor of NVIDIA; for which the AMD equivalent is a 'Compute Unit'. Practically, both CUDA cores and stream processors are closer to the individual vector lanes of a traditional CPU core. Given this minefield of confusing nomenclature we have instead opted to simply state the peak floating point capabilities of the hardware.

6.3. Results and discussion

By measuring the wall clock time required for PyFR to take 500 RK45[2R+] time-steps, and utilising the operation counts per time-step detailed in Fig. 3, one can calculated the sustained performance of PyFR in GFLOP/s when running with the meshes detailed in Section 5.3 with $\wp = 1, 2, 3, 4$.

Sustained performance of PyFR for the various hardware platforms is shown in Fig. 4. From the figure it is clear that the computational efficiency of PyFR increases with the polynomial order. This is consistent with higher order simulations having an increased compute intensity per degree of freedom. This additional intensity results in larger operator matrices that are better suited to the tiling schemes employed by BLAS libraries. The OpenCL implementation shipped by NVIDIA as part of CUDA only supports the use of 32-bit memory pointers. As such a single context is limited to 4 GiB of memory, *cf.* Table 3. It was therefore not possible to perform the third and fourth order simulations for either of the two meshes using the OpenCL backend with the K40c.

The Intel and AMD implementations of OpenCL, when used in conjunction with clBLAS, are only competitive with the C/OpenMP backend when $\wp = 1$ for the hexahedral mesh, and $\wp = 1, 2$ for the prism/tetrahedral mesh. This is also the case when comparing performance between the CUDA backend and the NVIDIA OpenCL backend on the K40c. Prior analysis by Witherden et al. [8] suggests that at these orders a reasonable proportion of the wall clock time will be spent in the bandwidth-bound point-wise kernels as opposed to DGEMM. On account of being bandwidth-bound such kernels do not

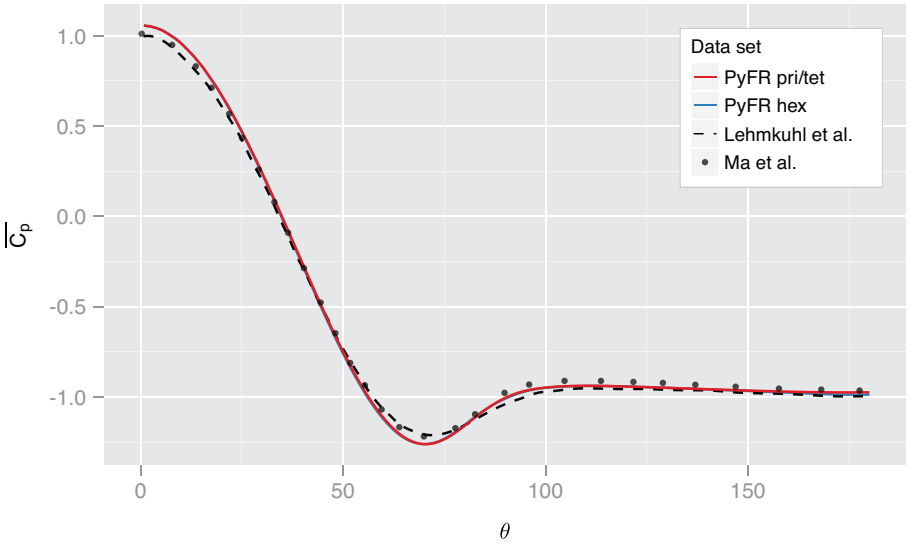


Fig. 9. Averaged pressure coefficient for Mode-H compared with the numerical results of Ma et al. [22] and Lehmkuhl et al. [26].

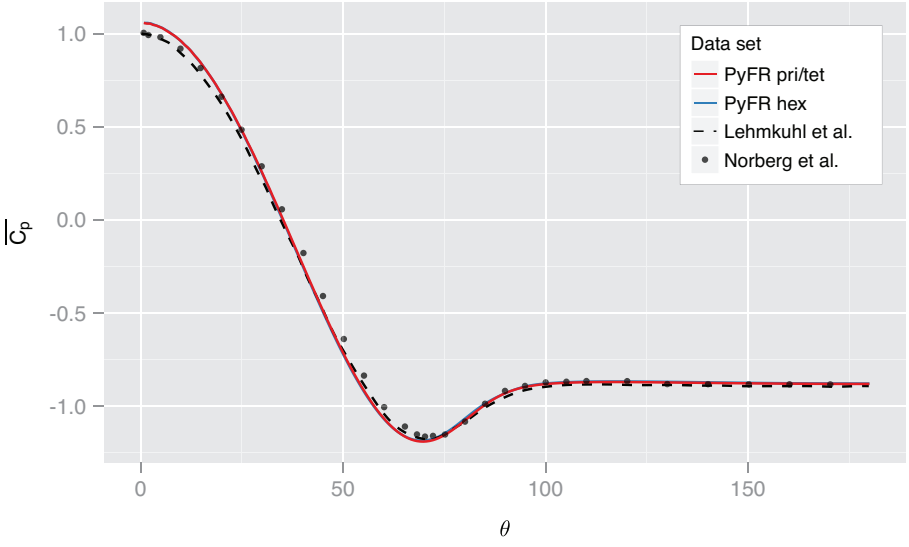


Fig. 10. Averaged pressure coefficient for Mode-L compared with the numerical results of Lehmkuhl et al. [26] and experimental results of of Norberg et al. [21] (from Kravchenko and Moin [24]).

extensively test the optimisation capabilities of the compiler. By the time $\wp = 4$ both implementations of OpenCL on the E5-2697 are delivering between one third and one quarter of the performance of the native backend. This highlights the lack of performance portability associated with OpenCL in this context, confirming our initial contention that, at the time of writing, performance portability can only be achieved effectively via native paradigms. Further, it also justifies the approach to multi-platform computing that has been adopted within PyFR.

Performance of the K40c culminates at 649 GFLOP/s for the $\wp = 4$ hexahedral mesh. This represents some 45% of the theoretical peak and 54% of the reference peak. By comparison the E5-2697 obtains 132 GFLOP/s for the same simulation equating to 47% and 57% of the theoretical and reference peaks, respectively. Performance does improve slightly to 140 GFLOP/s for the $\wp = 4$ prism/tetrahedral mesh, however. On this same mesh at $\wp = 4$ the W9100 can be seen to sustain 657 GFLOP/s of throughput. Although, in absolute terms, this observation represents the highest sustained rate of throughput it corresponds to just 25% of the theoretical peak. However, working in terms of realisable peaks, we find PyFR obtaining some 74% of the reference value.

Table 7
Comparison of quantitative values with experimental and DNS results.

	Mode-H		Mode-L	
	$-\overline{C_{pb}}$	$\theta_s/^\circ$	$-\overline{C_{pb}}$	$\theta_s/^\circ$
PyFR Pri/tet	0.974	87.13	0.882	86.90
PyFR Hex	0.987	88.28	0.880	87.71
Parnaudeau et al. [25]				88.00
Lehmkuhl et al. [26]	0.980	88.25	0.877	87.80
Norberg et al. [21, 24]			0.880	

The wall clock time required per degree of freedom (DOF) to evaluate $\nabla \cdot \mathbf{f}$ for each simulation can be seen in Table 5. The DOF count is inclusive of the factor of five arising from there being five distinct field variables at each solution point. This quantity can be used to evaluate the efficiency of PyFR relative to other codes. With the exception of

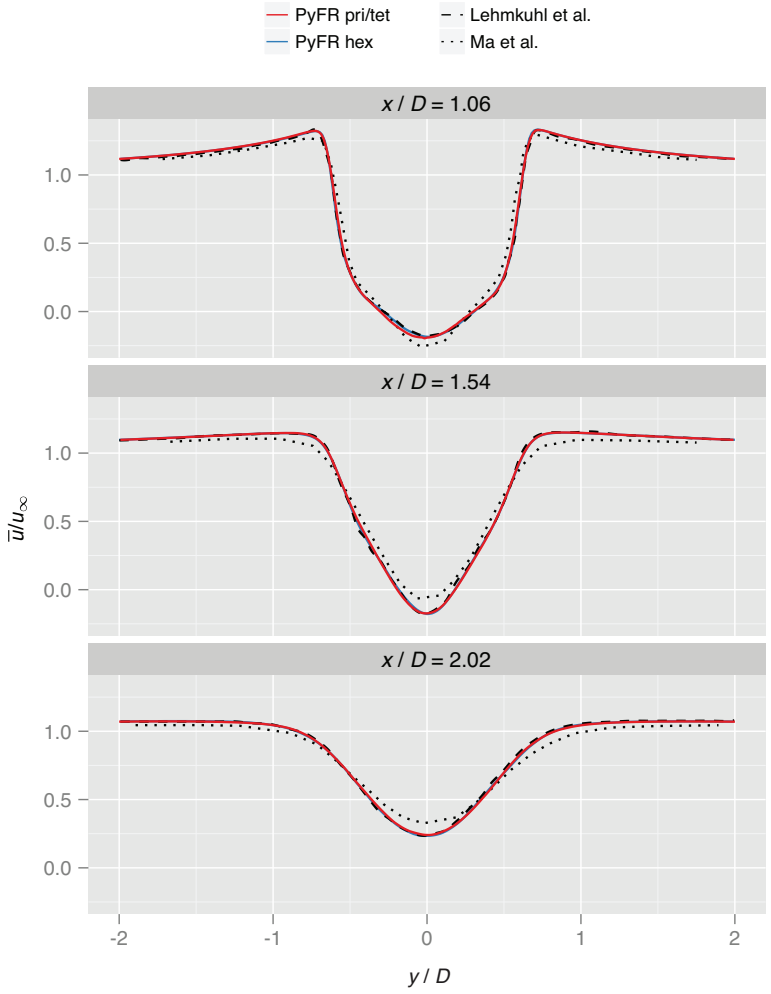


Fig. 11. Time-span-average stream-wise velocity profiles for Mode-H compared with the numerical results of Lehmkuhl et al. [26] and Ma et al. [22].

OpenCL on the E5-2697 we see that the time per DOF reaches a minima for the hexahedral mesh at $\wp = 3$. This shows that as $\wp$ is raised from one to three the increasing number of floating point operations required to update each DOF is being offset by the improving efficiency of PyFR. The pattern is similar for the prism/tetrahedral mesh except that for the E5-2697 (C/OpenMP) and the K40c (CUDA) the minima is at $\wp = 4$.

7. Multi-node heterogeneous performance

7.1. Overview

Having determined the performance characteristics of PyFR on various individual platforms, we will now investigate the ability of PyFR to undertake simulations on a multi-node heterogeneous system containing an Intel Xeon E5-2697 v2 CPU, an NVIDIA Tesla K40c GPU, and an AMD FirePro W9100 GPU. The experimental set up and methodology is the same as the single-node case.

7.2. Mesh partitioning

In order to distribute a simulation across the nodes of the heterogeneous system it is first necessary to partition the mesh. High quality partitions can be readily obtained using a graph partitioning package such as METIS [30] or SCOTCH [31].

When partitioning a mixed element mesh for a *homogeneous* cluster it is necessary to suitably weight each element type according to its computational cost. This cost depends both upon the platform on which PyFR is running and the order at which the simulation is being performed. In principle it is possible to measure this cost; however in practice the following set of weights have been found to give satisfactory results across most polynomial orders and platforms

$$\text{hex} : \text{pri} : \text{tet} = 3 : 2 : 1,$$

where larger numbers indicate a greater computational cost. One subtlety that arises here, is that from a graph partitioning standpoint there is no penalty associated with placing a sole vertex (element) of a given weight inside of a partition. Computationally, however, there is a very real penalty incurred from having just a single element of a certain type inside of the partition. It is therefore desirable to avoid mesh partitions where any one partition contains less than around a thousand elements of a given type. An exception is when a partition contains no elements of such a type—in which case zero overheads are incurred.

When partitioning a mesh with one type of element for a *heterogeneous* cluster it is necessary to weight the partition sizes in line with the performance characteristics of the hardware on each node. However, in the case of a mixed element mesh on a heterogeneous cluster the weight of an element is no longer static but rather depends on the partition that it is placed in—a significantly richer problem. Solving such a problem is currently beyond the capabilities of most graph partitioning packages. Accordingly, mixed element meshes that are partitioned for heterogeneous clusters often exhibit inferior load balancing than those partitioned for homogeneous systems. Moreover, for consistent performance it is necessary to dedicate a CPU core to each accelerator in the system. The amount of useful computation

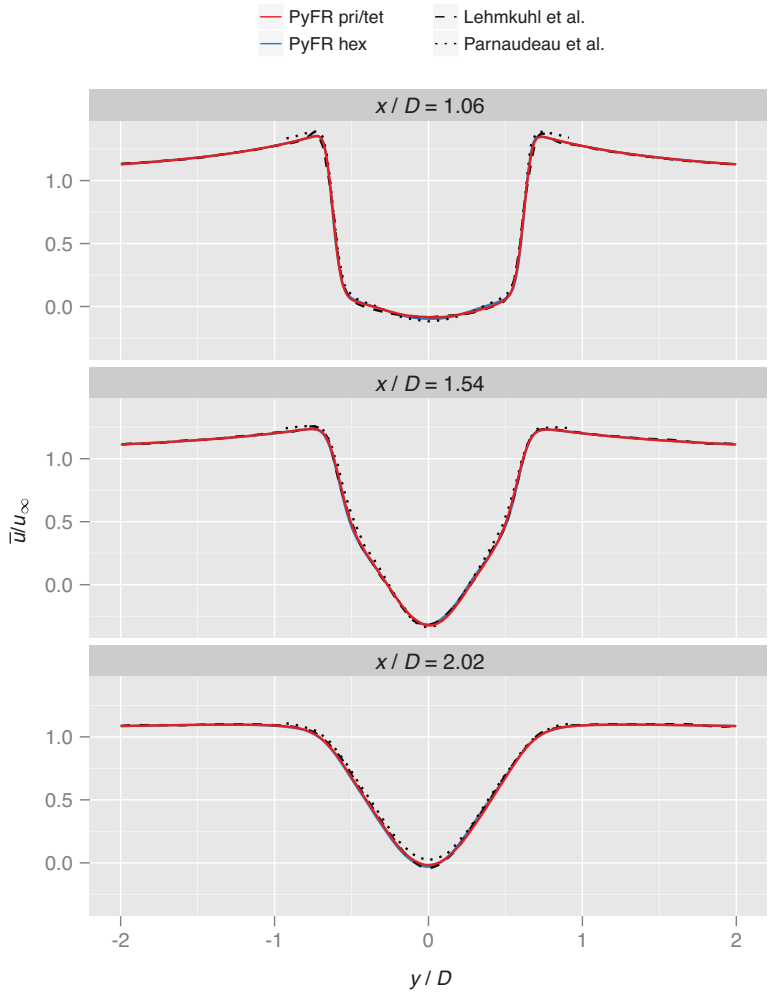


Fig. 12. Time-span-average stream-wise velocity profiles for Mode-L compared with the numerical results of Lehmkuhl et al. [26] and experimental results of Parnaudeau et al. [25].

that can be performed by the host CPU is therefore reduced in accordance with this.

Given the single-node performance numbers of Fig. 4 it comports to pair the E5-2697 with the C/OpenMP backend, the K40c with the CUDA backend, and the W9100 with the OpenCL backend, in order to achieve optimal performance. Employing the results of Fig. 4, in conjunction with some light experimentation, a set of partitioning weights were obtained and are tabulated in Table 6.

7.3. Results and discussion

Sustained performance of PyFR on the multi-node heterogeneous system for each of the meshes detailed in Section 5.3 with $\wp = 1, 2, 3, 4$ is shown in Fig. 5. Under the assumptions of perfect partitioning and scaling one would expect the sustained performance of the heterogeneous simulation to be equivalent to the sum of the E5-2697 (C/OpenMP), K40c (CUDA), and W9100 (OpenCL A) bars in Fig. 4. However, for reasons outlined in the preceding paragraphs these assumptions are unlikely to hold. Some of the available FLOP/s can therefore be considered as 'lost'. For the hexahedral mesh the fraction of lost FLOP/s varies from 22.5% when $\wp = 1$–8.7% in the case of $\wp = 4$. With the exception of $\wp = 1$ the fraction of lost FLOP/s are a few percent higher for the mixed mesh. This is understandable given the additional complexities associated with mixed mesh partitioning and can likely be improved upon by switching to order-dependent element weighting factors.

8. Accuracy

In this section we present instantaneous and time-span-averaged (henceforth referred to as averaged) results obtained using a cluster of 12 NVIDIA K20c GPUs at $\wp = 4$, the design resolution for both meshes. Both simulations are run for 1000 convective times, allowing the flow to fluctuate between Mode-H and Mode-L as identified by Lehmkuhl et al. [22, 26]. A moving window time-average with a width of 100 convective times is used to extract both modes from the long-period simulation. This yields four datasets including both Mode-H and Mode-L for both the hexahedral and prism/tetrahedral meshes. Both modes are then compared with results from previous experimental and numerical studies, where either one or both of the modes were observed [21, 22, 25, 26].

Instantaneous surfaces of iso-density are shown in Fig. 6 for both simulations at similar phases of the shedding cycle. We observe laminar flow at the leading edge of the cylinder for both test cases, turbulent transition near the separation points, and fully turbulent flow in the wake region. These are the characteristic features of the shear-layer transition regime, as described by Williamson [20]. The wake is composed of large vortices, alternately shedding off of the upper and lower surfaces of the cylinder, and smaller scale turbulent structures.

Plots of the averaged stream-wise wake profiles are shown in Figs. 7 and 8 for Mode-H and Mode-L, respectively. Both the hexahedral and prism/tetrahedral meshes show excellent agreement with the numerical results of Lehmkuhl et al. [26] for both modes and with the experimental results of Parnaudeau et al. [25], which is available

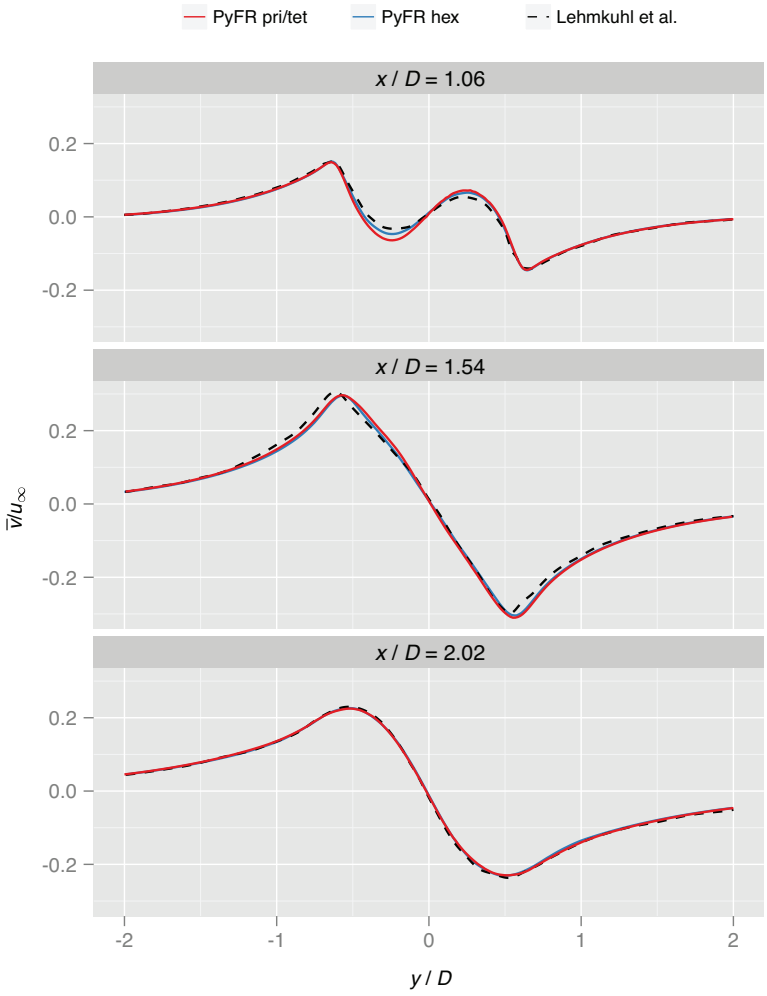


Fig. 13. Time-span-average cross-stream velocity profiles for Mode-H compared with the numerical results of Lehmkuhl et al. [26].

for Mode-L. The Mode-H cases exhibit relatively shorter separation bubbles and the Mode-L cases have characteristic inflection points in the wake profile near x/D $\approx$ 1.

Plots of the averaged pressure coefficient $\overline{C_p}$ on the surface of the cylinder are shown in Figs. 9 and 10 for both extracted modes and both meshes. The Mode-H results are shown alongside the Mode-H numerical results of Lehmkuhl et al. [26] and the results from Case I of Ma et al. [22]. The Mode-L results are shown alongside the Mode-L numerical results of Lehmkuhl et al. [26] and the experimental results of Norberg et al. at a similar $Re = 4020$ [21], which were extracted from Kravchenko and Moin [24]. Both modes have similar pressure coefficient distributions at the leading face of the cylinder, while the Mode-H case has stronger suction on the trailing face adjacent to the separation bubble. Both modes extracted using both meshes show excellent agreement with their corresponding reference data sets.

The averaged pressure coefficient at the base of the cylinder $\overline{C_{pb}}$, and the averaged separation angle θ_s measured from the leading stagnation point are tabulated in Table 7 for both modes and meshes. These are shown along with measurements from the experimental results of Norberg et al. [21], experimental data from Parnaudeau et al. [25], and DNS data from Lehmkuhl et al. [26] for both modes. Both measured quantities agree well with the reference data sets for both modes and meshes. The difference in separation angle is less than $\sim 1^\circ$ between the current and reference results. The pressure coefficient at base of the cylinder shows that the high-energy Mode-H case has stronger recirculation in the wake, characterised by greater suction at the wall adjacent to the recirculation bubble.

Plots of averaged stream-wise velocity at $x/D = 1.06$, 1.54, and 2.02 are shown in Figs. 11 and 12 for the Mode-H and Mode-L simulations, respectively. These results are shown alongside the experimental results of Parnaudeau et al. [25] for Mode-L, the numerical results of Ma et al. [22] for Mode-H, and the DNS results of Lehmkuhl et al. [26] for both modes. Both the prism/tetrahedral and hexahedral mesh simulations show the characteristic V-shaped velocity profile for Mode-H at $x/D = 1.06$. They also show the characteristic U-shaped profile for Mode-L, also at $x/D = 1.06$. Both modes on both meshes agree well with both their corresponding reference data sets. Plots of averaged cross-wise velocity at $x/D = 1.06$, 1.54, and 2.02 are shown in Figs. 13 and 14, also for the Mode-H and Mode-L simulations. These cross-wise velocity profiles also show excellent agreement with their corresponding reference data sets.

9. Conclusions

We have detailed the extension of PyFR to run on mixed element meshes, and a range of hardware platforms, including heterogeneous multi-node systems. Performance of our implementation was benchmarked using pure hexahedral and mixed prismatic-tetrahedral meshes of the void space around a circular cylinder. Specifically, for each mesh performance was assessed at various orders of accuracy on three different hardware platforms; an NVIDIA Tesla K40c GPU, an Intel Xeon E5-2697 v2 CPU, and an AMD FirePro W9100 GPU. Performance was then assessed on a heterogeneous multi-node system constructed from a mix of the

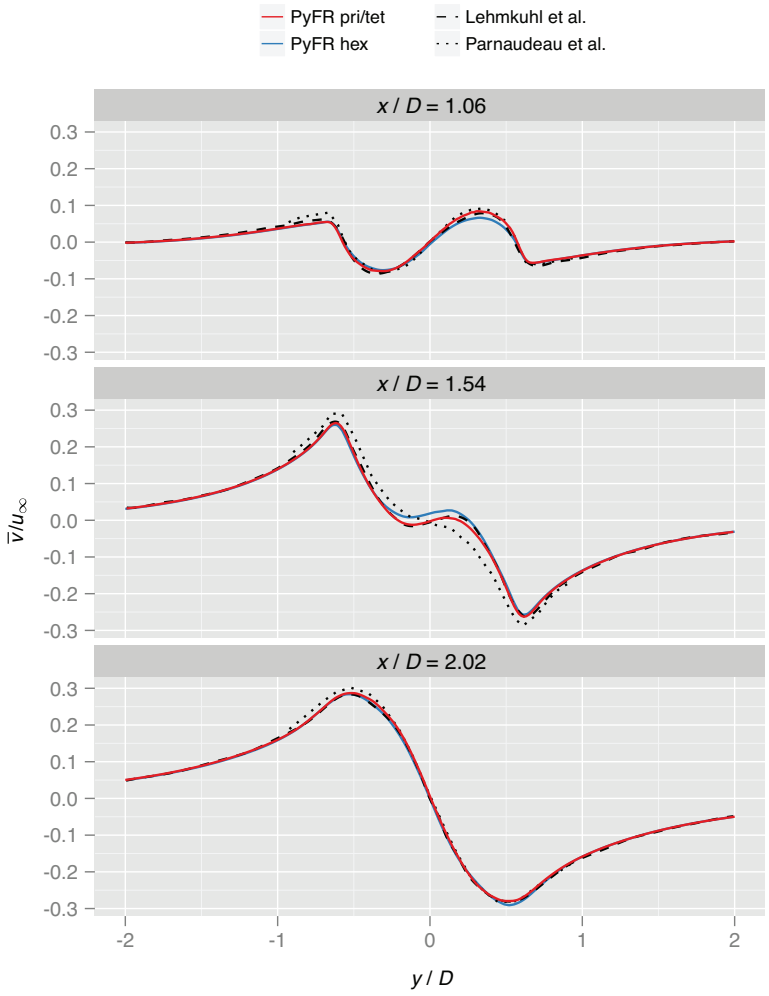


Fig. 14. Time-span-average cross-stream velocity profiles for Mode-L compared with the numerical results of Lehmkuhl et al. [26] and experimental results of Parnaudeau et al. [25].

aforementioned hardware. Results demonstrated that PyFR achieves performance portability across various hardware platforms. In particular, the ability of PyFR to target individual platforms with their 'native' language leads to significantly enhanced performance *cf.* targeting each platform with OpenCL alone. PyFR was also found to be performant on the heterogeneous multi-node system, achieving a significant fraction of the available FLOP/s. Finally, each mesh was used to undertake nominally fifth-order accurate long-time simulations of unsteady flow over a cylinder at a Reynolds number of 3900 using a cluster of NVIDIA K20c GPUs. Long-time dynamics of the wake were studied in detail, and results were found to be in excellent agreement with previous experimental/numerical data.

Acknowledgements

The authors would like to thank the Engineering and Physical Sciences Research Council for their support via a Doctoral Training Grant, an Early Career Fellowship (EP/K027379/1), and the Hyper Flux project (EP/M50676X/1). The authors would also like to thank AMD, Intel, and NVIDIA for hardware donations.

Supplementary material

Data Statement: Data relating to the results in this manuscript can be downloaded as Electronic Supplementary Material under a CC-BY-NC-ND 4.0 license. Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.compfluid.2015.07.016.

References

[1] ReedWH, HillTR. Triangular mesh methods for the neutron transport equation. 1973. Technical report LA-UR-73-479, Los Alamos Scientific Laboratory.
[2] Kopriva David A, Kolias John H. A conservative staggered-grid Chebyshev multidomain method for compressible flows. J Comput Phys 1996;125(1):244–61.
[3] Sun Yuzhi, Wang Zhi Jian, Liu Yen. High-order multidomain spectral difference method for the Navier-Stokes equations on unstructured hexahedral grids. Commun Comput Phys 2007;2(2):310–33.
[4] Huynh HT. A flux reconstruction approach to high-order schemes including discontinuous Galerkin methods. AIAA Pap 2007;4079:2007.
[5] Hesthaven Jan S, Warburton Tim. Nodal discontinuous Galerkin methods: algorithms, analysis, and applications, 54.. New York: Springer Verlag; 2008.
[6] Klöckner Andreas, Warburton Tim, Bridge Jeff, Hesthaven Jan S. Nodal discontinuous Galerkin methods on graphics processors. J Comput Phys 2009;228(21):7863–82.
[7] Castonguay Patrice, Williams David M, Vincent Peter E, Lopez Manuel, Jameson Antony. On the development of a high-order, multi-gpu enabled, compressible viscous flow solver for mixed unstructured grids. AIAA Pap 2011;3229:2011.
[8] Witherden FD, Farrington AM, Vincent PE. PyFR: an open source framework for solving advection-diffusion type problems on streaming architectures using the flux reconstruction approach. Comput Phys Commun 2014;185(11):3028–40.
[9] Vincent PE, Castonguay P, Jameson A. A new class of high-order energy stable flux reconstruction schemes. J Sci Comput 2011;47(1):50–72.
[10] Castonguay P, Vincent P, Jameson A. A new class of high-order energy stable flux reconstruction schemes for triangular elements. J Sci Comput 2011;51(1):224–56.
[11] Williams DM, Jameson A. Energy stable flux reconstruction schemes for advection-diffusion problems on tetrahedra. J Sci Comput 2013;59(3):721–59.

[12] Goto Kazushige, Geijn Robert A. Anatomy of high-performance matrix multiplication. ACM Trans Math Softw 2008;34(3):12.
[13] BayerMichael. Mako: templates for python. 2013.
[14] Klöckner Andreas, Pinto Nicolas, Lee Yunsup, Catanzaro Bryan, Ivanov Paul, Fasih Ahmed. Pycuda and pyopencl: a scripting-based approach to gpu run-time code generation. Parallel Comput 2012;38(3):157–74.
[15] Vermeire BC, Nadarajah S. Adaptive IMEX time-stepping for ILES using the correction procedure via reconstruction scheme. AIAA Pap 2013;2687:2013.
[16] Vermeire BC, Nadarajah S. Adaptive IMEX schemes for high-order unstructured methods. J Comput Phys 2015;280:261–86.
[17] RoshkoAnatol. On the development of turbulent wakes from vortex streets. 1953. Technical report no. NACA TR 1191, California Institute of Technology.
[18] Bloor MS. The transition to turbulence in the wake of a circular cylinder. J Fluid Mech 1964;19:290–304.
[19] Williamson CHK. The existence of two stages in the transition to three dimensionality of a cylinder wake. Phys Fluids 1988;31:3165–8.
[20] Williamson CHK. Vortex dynamics in the cylinder wake. Ann Rev Fluid Mech 1996;28:477–539.
[21] Norberg C. Ldv measurements in the near wake of a circular cylinder. Int J Numer Methods Fluids 1998;28(9):1281–302.
[22] Ma X, Karamanos GS, Karniadakis GE. Dynamics and low-dimensionality of a turbulent near wake. J Fluid Mech 2000;310:29–65.
[23] Breuer M. Large eddy simulation of the subcritical flow past a circular cylinder. Int J Numer Methods Fluids 1998;28(9):1281–302.
[24] Kravchenko AG, Moin P. Numerical studies of flow over a circular cylinder at $re_d = 3900$. Phys Fluids 2000;12:403–17.
[25] Parnaudeau Philippe, Carlier Johan, Heitz Dominique, Lamballais Eric. Experimental and numerical studies of the flow over a circular cylinder at reynolds number 3900. Phys Fluids 2008;20(8):085101–085101-14.
[26] Lehmkuhl O, Rodriguez I, Borrell R, Oliva A. Low-frequency unsteadiness in the vortex formation region of a circular cylinder. Phys Fluids 2013;25(8):3165–8.
[27] Kennedy Christopher A, Carpenter Mark H, Lewis RMichael. Low-storage, explicit runge–kutta schemes for the compressible navier–stokes equations. Appl Numer Math 2000;35(3):177–219.
[28] Vermeire BC, Cagnone JS, Nadarajah S. ILES using the correction procedure via reconstruction scheme. AIAA Pap 2013;1001:2013.
[29] Vermeire BC, Nadarajah S, Tucker PG. Canonical test cases for high-order unstructured implicit large eddy simulation. AIAA Pap 2014;0935:2014.
[30] Karypis George, Kumar Vipin. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J Sci Comput 1998;20(1):359–92.
[31] Pellegrini François, Roman Jean. Scotch: a software package for static mapping by dual recursive bipartitioning of process and architecture graphs. High-Performance Computing and Networking. Springer; 1996. p. 493–8.