Improvement of the computational performance of a parallel unstructured WENO finite volume CFD code for Implicit Large Eddy Simulation

Panagiotis Tsoutsanis*, Antonis F. Antoniadis, Karl W. Jenkins
Centre for Computational Engineering Sciences, Cranfield University, Cranfield, MK43 0AL, United Kingdom

A R T I C L E   I N F O
Article history:
Received 22 September 2017
Revised 29 January 2018
Accepted 1 March 2018
Available online 5 March 2018

A B S T R A C T
In this paper the assessment and the enhancement of the computational performance of a high-order finite volume CFD code is presented. Weighted Essentially Non-Oscillatory (WENO) schemes are considered to be from the most computationally expensive numerical frameworks, in the context of high-resolution schemes particularly on hybrid unstructured grids. The focus of this study is to assess the computational bottlenecks of the solver for the WENO schemes for Implicit Large Eddy Simulation (ILES) and optimise the performance and efficiency through a series of code modifications e.g. formula rewriting, reduction of number operations, inclusion of linear systems libraries, non-blocking communications amongst others. The code is assessed on five different HPC systems; significant speed-up is achieved ranging from 1.5 to 8.5, with very high-order schemes benefiting the most. Good scalability is also obtained up to $10^3$ number of cores, demonstrating viability and affordability of WENO type schemes for scale resolving simulations.

1. Introduction

Numerical methods of the high-order family were prohibited a decade ago in terms of great computational effort for large scale, moderate to high -Reynolds number cases; now these methods are being increasingly adopted for various industrial applications e.g. acoustics, combustion, rotating wings and turbo machinery and are available in commercial solver packages.

High-order numerical methods can provide improved accuracy at a reduced computational cost for transient CFD applications compared to second-order schemes. There is a wealth of high-order numerical methods for unstructured meshes developed across different frameworks including the finite volume (FV) [1–10], the Discontinuous Galerkin (DG) [11–16], the Spectral Finite Volume (SFV) methods [17–22] and the flux reconstruction scheme (FR) [23–28].

High-order methods such as WENO or DG with explicit time-stepping schemes for time advancement of time-dependent simulations have demonstrated to be attractive solutions for scale resolving simulations. The popularity of these solutions is mainly attributed to the favourable scalability of these solvers, where higher computational over communication load ratios can be obtained compared with lower-order spatial schemes.

WENO type schemes are considered to be computationally expensive particularly for unstructured meshes and for very high-order accuracy (>5th order). This is mainly due to their non-uniform stencils construction, cumbersome mesh partitioning, complex communication strategies; however, their enhanced accuracy, robustness, conservation properties, flexibility for multiphysics model extensions and ability to provide spurious-free solutions counterbalances the aforementioned drawback and makes them attractive to a wider set of applications. The method presented in this work is based on a series of works published over the past years; in references [7,29] the WENO framework is employed to discretise the Euler equations on hybrid unstructured grids; the extension of the method for time dependent laminar, transitional and turbulent flows are presented in [9] and for industrial scale applications and high-order discretisation of the RANS equations in [10]. The present higher-order WENO methods are producing significantly smaller errors for a series of test problems, compared to a standard 2nd-order Monotonic Upwind Scheme for Conservation Laws (MUSCL) scheme at approximately similar computing times [9,30]. Additionally they offer superior

* Corresponding author.
E-mail addresses: panagiotis.tsoutsanis@cranfield.ac.uk (P. Tsoutsanis), a.f.antoniadis@cranfield.ac.uk (A.F. Antoniadis), k.w.jenkins@cranfield.ac.uk (K.W. Jenkins).

https://doi.org/10.1016/j.compfluid.2018.03.012
0045-7930/Crown Copyright © 2018 Published by Elsevier Ltd. This is an open access article under the CC BY license. (http://creativecommons.org/licenses/by/4.0/)
scalloplity compared to lower-order schemes since their ratio of
computation over communication time is larger than a MUSCL
2nd-order scheme [9].

Writing an efficient program code of WENO type method for
mixed-element unstructured grids can be a daunting task. The
complexity of the scheme and the explicit graph-type data for-
mat inherited by the unstructured framework results to several
challenges: memory size limitation, random memory access pat-
terns and computational cost amongst others. Major changes in
the types of hardware utilised for scientific computing such as graphi-

cal processing units (GPUs), lower clock speeds, and many-core ar-chitecture processors and co-processors set additional challenges
to the solver’s efficiency. CFD codes will require radical changes
in order to utilise the full potential of these new architectures.
A possible and attractive direction would be the extension of current
CFD solvers is to incorporate high-order and even very high-order
methods which would be tuned to the extreme levels of parallel-
ism of the new hardware. Additionally, that may imply that new
implementation techniques of numerical methods need to be de-
veloped, taking into account the changes in hardware in terms of
the different characteristics from the hardware architectures of
the previous decades.

The prime objective of this current work is to assess the com-
putational characteristics of WENO finite volume schemes for un-
structured meshes for time dependent explicit Implicit Large Eddy
Simulations (ILES). This is accomplished by identifying the code
sections and subroutines that have the highest computational cost
and memory footprint. The workflow pattern of unstructured fi-
nite volume CFD codes, includes operations characterised by non-
coalescent memory accessing patterns, conditional branching for
fluxes approximations, cell-based fine-grained adaptivity of the nu-
merical methods. These type of operations prevents the CFD code
from belonging to the embarrassingly parallel framework of ap-
lications.

As it is demonstrated in this work, the associated cost of these
operations is insignificant compared to the cost of the numer-
ous polynomial reconstructions that take place within the WENO
framework. In this study we are interested in reducing the computa-
tional cost of WENO schemes, although other techniques for im-
proving their compactness could be explored similarly to Dumbser
et al. [31], we are pursuing this through a series of enhancements
in the execution of the algorithms and operations rather altering
the numerical methodology. These enhancements are tested across
five different HPC architectures including a Intel Xeon Phi Knights-
Landing manycore processor, and to the best of our knowledge this
is the first time that very high-order finite volume WENO schemes
for unstructured meshes, have been assessed, optimised and de-
ployed for ILES of turbulent flows in this type of manycore ar-
chitectures.

The rest of the paper is organized as follows. Section 2 is de-
icated to the description of the general numerical framework, the
implementation of these algorithms along with a brief description
of the CFD solver code name UCNS3D (Unstructured Compressible
Navier–Stokes 3D). In Section 3 we discuss the performance ob-
tained with the original implementation of the code, identifying
the most expensive processes, the optimisations explored for im-
proving the computational performance, and the performance of
the revised CFD code across five different HPC clusters for various
schemes. Finally the conclusions of the present study are outlined
in the last section.

2. Methodology

An overview of the governing equations, spatial and temporal
discretisation schemes is presented; followed by the implementa-
tion of the schemes, where their computational and commu-
nication patterns are discussed in detail.

2.1. Governing equations

The compressible Navier–Stokes equations are considered, writ-
ten in conservative form as:

\[
\frac{\partial U(x,t)}{\partial t} + \nabla \cdot \mathbf{F}(U, \nabla U) = 0,
\]

where \( U \) is the vector of the conserved mean flow variables, and \( \mathbf{F} \)
is the non-linear flux tensor given by:

\[
\mathbf{F}(U, \nabla U) = \left( \begin{array}{c}
\bar{u}^T \\
\bar{u}^T \rho + \sigma(U, \nabla U)
\end{array} \right) \left( \begin{array}{c}
\bar{u} \\
\bar{u} \nabla U
\end{array} \right).
\]

where the stress tensor \( \sigma \) is given by:

\[
\sigma = \left( p + \frac{2}{3} \mu \nabla u \right) I - \mu (\nabla u + \nabla u^T).
\]

In the above equations, \( \rho \) is the density and ideal gas is
assumed where the total energy is given by \( E = p/\gamma - 1 +
(1/2) \rho (u^2 + v^2 + w^2) \), where \( p \) is the pressure, \( \gamma \) is the ra-
tio of specific heats for air at normal atmospheric conditions; The
laminar viscosity is related to the temperature through Sutherland
law:

\[
\mu_2 = \left( \frac{T}{T_0} \right) \frac{\mu_0}{\frac{T_0}{T} + S}.
\]

\( S \) is the Sutherland temperature and the subscript 0 denotes a
reference state for the corresponding variables.

2.2. Numerical framework

The discretisation in a domain \( \Omega \) is achieved by combining
conforming arbitrary shaped elements of volume \( |V| \). Integrating
Eq. (1) over a mesh element using the finite volume formulation
the ordinary differential equation is obtained as follows:

\[
\frac{dU_i}{dt} = \frac{1}{|V|} \sum_{l=1}^{N_l} \sum_{a=1}^{N_a} \sum_{e=1}^{N_e} \int_{V_e} \mathbf{F}(\mathbf{x}_e, t) \cdot \nabla U(\mathbf{x}_e, t) \alpha_e |A_l|,\]

where \( U_i \) is the volume averaged conserved variable vector, \( N_l \)
is the number of faces per element, \( N_a \) is the number of quadrature
points used for approximating the surface integrals, \( |A_l| \) is the sur-
f ace area of the corresponding face, and \( \alpha \) corresponds to differ-
ent Gaussian integration points \( \mathbf{x}_e \) and weights \( \omega_e \) over the face. The
weight and distribution of the quadrature points depend upon the
order of the Gaussian quadrature integration rule employed, and
for the present study suitable rules for the employed polynomial
order are used. The interface fluxes are computed based on the
boundary extrapolated reconstructed values, which are obtained by
a polynomial reconstruction from element-averaged data. The fol-
lowing section describes the methodology adopted for the spatial
discretisation.

2.2.1. High-order finite volume k-exact least-square reconstruction

The main objective of the reconstruction process is to build a
high-order polynomial \( p(x, y, z) \) of arbitrary order \( r \), for each con-
sidered element \( V_l \) that has the same average as a general quantity
\( U_l \) this yields:

\[
U_l = \frac{1}{|V_l|} \int_{V_l} U(x, y, z) dV = \frac{1}{|V_l|} \int_{V_l} \int p_i(x, y, z) dV.
\]
The present reconstruction algorithm is based upon the approaches of [7,29], that has been successful applied to smooth and discontinuous flow problems [9,30,32–34]. The reconstruction is carried out in a transformed system of coordinates in order to minimize scaling effects that appear in stencils consisting of elements of different size as shown in Fig. 1 and in order to improve the condition number of the system of equations. The reader is referred to [7,29,35] and references therein for more details of the approach adopted. The transformation is achieved by decomposing each element into tetrahedral elements. This transformation is only computed once at the initialization phase of the simulation since the mesh does not change with time for the present study. Using an inverse mapping the considered element $V_i$ can be transformed to the element $V_i'$ in the reference co-ordinate system.

The spatial average of the conserved variable $U_i$ does not change during coordinate transformation

$$U_i = \frac{1}{|V_i|} \int_{V_i} U(x,y,z) \, dV \equiv \frac{1}{|V_i|} \int_{V_i} U(\xi, \eta) \, d\xi d\eta d\zeta.$$  

To perform the reconstruction on the target element $V_i$, we construct the central reconstruction stencil $S$ by recursively adding neighbouring elements, consisting of $M+1$ elements, including the target element $V_i$ as shown in Fig. 1.

$$S = \bigcup_{m=0}^{M} V_m,$$  

where the index $m$ refers to the local numbering of the elements in the stencil, with the element with index 0 being the considered element $i$. The $r$th order reconstruction polynomial at the transformed element $V_m'$ is sought as an expansion over local polynomial basis functions $\phi_k(\xi, \eta, \zeta)$ given by:

$$p(\xi, \eta, \zeta) = \sum_{k=0}^{K} a_k \phi_k(\xi, \eta, \zeta) = U_0 + \sum_{k=1}^{K} a_k \phi_k(\xi, \eta, \zeta),$$  

where $U_0$ corresponds to the general quantity at the considered element $i$, and $\xi$, $\eta$, $\zeta$ are the coordinates in the reference system. $a_k$ are the degrees of freedom and the upper index in the summation of expansion $K$ corresponds to the number of the degrees of freedom and is related to the order of the polynomial $r$ by $K = \frac{1}{2}(r+1)(r+2)(r+3) - 1$. For computing the degrees of freedom $a_k$, a minimum of $K$ element are required within the stencil, in addition to the target element. For improving the robustness of the algorithm we use $M = 2 \cdot K$ as described in [7,29,30,35].

To find the unknown degrees of freedom $a_k$ for each element $m$ from the stencil the cell average of the reconstruction polynomial $p(\xi, \eta, \zeta)$ must be equal to the element average of the solution $U_m$:

$$\int_{V_m} p(\xi, \eta, \zeta) \, d\xi d\eta d\zeta = |V_m'/U_0| + \sum_{k=1}^{K} a_k \phi_k \int_{V_m} d\xi d\eta d\zeta$$

$$= |V_m'/U_m|, \quad m = 1, \ldots, M.$$

(11)

Where $V_m'$ is the volume of the element $m$ in the stencil, in the transformed coordinate system. Additionally the basis functions $\phi_k$ will satisfy the constraint of Eq. (7) irrespective of the values of degrees of freedom. The basis functions $\phi_k$ for all the elements in the stencil are defined as follows:

$$\phi_k(\xi, \eta, \zeta) \equiv \psi_k(\xi, \eta, \zeta) - \frac{1}{|V_0'|} \int_{V_0} \psi_k(\xi, \eta, \zeta) \, d\xi d\eta d\zeta, \quad k = 1, \ldots, K.$$

(12)

In the present study the basis functions are based on the Legendre type polynomial. Denoting the integrals of the basis function $k$ over the considered element $m$ in the stencil, and the vector of right-hand side by $A_{mk}$ and $b$ respectively as given by:

$$A_{mk} = \int_{V_m} \phi_k \, d\xi d\eta d\zeta, \quad b_m = |V_m'/(U_m - U_0)|.$$

(13)

we rewrite the equations for degrees of freedom $a_k$ in a matrix form as

$$\sum_{k=1}^{K} A_{mk} a_k = b_m, \quad m = 1, \ldots, M.$$

(14)

The complete linear system is given as:

$$\left[ \Phi_{1,1}(\xi, \eta, \zeta) \Phi_{1,2}(\xi, \eta, \zeta) \cdots \Phi_{1,K}(\xi, \eta, \zeta) \right] \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_K \end{bmatrix} = \begin{bmatrix} |V_1'|/(U_1 - U_0) \\ |V_2'|/(U_2 - U_0) \\ \vdots \\ |V_M'|/(U_M - U_0) \end{bmatrix}.$$

(15)

Where $\Phi_{m,k}$ corresponds to the integral of the basis function $\phi_k$ for element $m$. The matrix $A_{mk}$ is purely based on the geometry of the stencil’s elements, which remains unchanged during the simulation. On the other hand the information on the right-hand vector

Fig. 1. Examples of central stencil and directional stencils for a 5th-order WENO scheme on a 3D tetrahedral mesh, with the considered element in red colour. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)
$b_m$ is dependent upon the solution, and the volume of each of the element in the stencil in the transformed coordinate system. The matrix $A^T_{km}A_{mk}$ is invertible and the Eq. (14), can be rewritten in the following form:

$$a_k = (A^T_{km}A_{mk})^{-1}A^T_{km}b_m = A^T_{km}b_m,$$  \hspace{1cm} (16)

where the matrix $A^T_{km}$ corresponds to the Moore–Penrose pseudo-inverse of $A_{mk}$. The polynomial coefficients are therefore obtained through a matrix vector multiplication of $A^T_{km}$ with $b_m$. A QR decomposition based on Householder transformation is employed where the pseudo-inverse is decomposed in an orthogonal matrix $Q$ and an upper triangular one $R$, and the inverse of $R^T R$ is obtained by a forward-substitution followed by a backward-substitution, this yields:

$$A^T_{km} = ((QR)^T (QR))^{-1}A^T_{km} = (R^T R)^{-1}A^T_{km}.$$  \hspace{1cm} (17)

The main memory requirements of the least-square reconstruction are associated with the storage of the pseudo-inverse matrix $A^T_{km}$ that is only computed once at the initialisation stage of the simulation.

2.2.2. WENO scheme

The WENO scheme employed in this study, utilises a non-linear combination of various reconstruction polynomials from the central stencil and the directional stencils; which are shown in Fig. 1, where each polynomial is weighted according to the smoothness of its solution, and it is based on the approaches of [7,35]. The polynomials are given as:

$$p^{\text{weno}}_i = \sum_{s=1}^{s_i} \omega_m p_s(\xi, \eta, \zeta),$$  \hspace{1cm} (18)

where $s_i$ is the total number of WENO stencils. Substituting back to Eq. (10) for $p_i(\xi, \eta, \zeta)$, we obtain the following expression:

$$p_i(\xi, \eta, \zeta) = \sum_{k=0}^{K} q^s_k \phi_k(\xi, \eta, \zeta).$$  \hspace{1cm} (19)

Using the condition that the sum of all weights is unity, yields:

$$p^{\text{weno}}_i = U_0 + \sum_{k=1}^{K} \tilde{\omega}_k \phi_k(\xi, \eta, \zeta) \equiv U_0 + \sum_{k=1}^{K} \delta_k \phi_k(\xi, \eta, \zeta),$$  \hspace{1cm} (20)

where $\delta_k$ are the reconstructed degrees of freedom; and the non-linear weight $\omega_m$ is defined as:

$$\omega_m = \sum_{k=1}^{s_i} \tilde{\omega}_k \delta_k \quad \text{where} \quad \tilde{\omega}_k = \frac{\lambda_m}{(\epsilon + S\lambda_m)^\beta},$$  \hspace{1cm} (21)

The smoothness indicator $S\lambda_m$ is given by:

$$S\lambda_m = \sum_{1 \leq |\beta| \leq |\gamma|} \int_{V_i} \left(D^\beta p_i(\xi, \eta, \zeta)\right)^2 (d\xi, d\eta, d\zeta),$$  \hspace{1cm} (22)

where $\beta$ is a multi-index, $r$ is the polynomial’s order, $\lambda_m$ is the linear weight. The central stencil is assigned a large linear weight of $\lambda_1 = 1000$ and a value to prevent division by zero of $\epsilon = 10^{-6}$ is used and $D$ is the derivative operator. The reader is referred to [7,29] for the definition of geometrical sectors and a detailed explanation of the different set of geometrical conditions. The smoothness indicator is a quadratic function of the degrees of freedom ($\alpha_k$) and Eq. (22) can be rewritten as:

$$S\lambda_m = \sum_{k=1}^{K} \alpha_k \left( \sum_{q=1}^{Q} \omega_{iq} \alpha_q^2 \right),$$  \hspace{1cm} (23)

where the oscillation indication matrix $O\lambda_{iq}$ is given by:

$$O\lambda_{iq} = \sum_{1 \leq |\beta| \leq |\gamma|} \int_{V_i} \left(D^\beta \phi_i(\xi, \eta, \zeta)\right)(d\xi, d\eta, d\zeta).$$  \hspace{1cm} (24)

and can be easily precomputed and stored at the initialisation stage of the simulation. Additionally the WENO reconstruction is carried out with respect to the characteristic variables for the present study. An average value in the direction normal to each element face is defined as $U_{nx}$, the conserved vector $U_{L,n}$, and the $U_{R,n}$ correspond to the left and right states at the cell interface $I$ in the normal direction. Denoting the right and left eigenvectors of the convective Jacobian matrix on the normal direction to the cell interface as $R_I$ and $L_I$ respectively, both of which are calculated based on the average cell state $U_n$. Then the projections of the degrees of freedom for each stencil $s$, $a_k^s$, are projected to the characteristic variables as:

$$B_{I,n}^k = L_I a_k^s, \quad s = 1, \ldots, 6; \quad k = 0, \ldots, K,$$  \hspace{1cm} (25)

where the WENO reconstruction is applied to each characteristic variable. Then the WENO modified degrees of freedom $B_{I,n}$ are transformed back to the conserved formulation by the right eigenvector $R_I$, as follows:

$$\tilde{a}_k = R_I B_{I,n}, \quad k = 0, \ldots, K,$$  \hspace{1cm} (26)

with the final WENO reconstruction polynomial for face $I$ being given by:

$$P_I(\xi, \eta, \zeta) = U_0 + \sum_{k=1}^{K} \tilde{a}_k \phi_k(\xi, \eta, \zeta).$$  \hspace{1cm} (27)

The WENO reconstruction with respect to characteristic variables is more computationally expensive as opposed to the conservative variables, since the WENO weights are different for each element’s interface, and due to the forward and backward projection of the degrees of freedom to characteristic variables.

2.2.3. Gradients for viscous terms

For the evaluation of the viscous fluxes the boundary extrapolated values for velocity $\nabla u_{i,a}$ and temperature $\nabla T_{i,a}$ for element $i$, for face $l$ and Gaussian quadrature point $a$ are required. They are computed by multiplying the transpose of the inverse Jacobian $(J^{-1})^T$ with the first order derivatives of the polynomial of the central stencil. For the gradient of $u$-velocity $\nabla u_{i,a}$ this is computed as follows:

$$\begin{pmatrix} \partial u \\ \partial w \\ \partial \phi \\ \partial v \end{pmatrix} = (J^{-1})^T \begin{pmatrix} \partial \phi(\xi, \eta, \zeta) \\ \partial \eta \\ \partial \zeta \\ \partial \xi \end{pmatrix}. \hspace{1cm} (28)$$

For the evaluation of the velocity and temperature gradients at the no-slip boundaries, the Dirichlet and Neumann boundary conditions are satisfied respectively through the least-square reconstruction similarly to [6,9] (Fig. 3).

2.2.4. Numerical fluxes

For the evaluation of the convective fluxes the approximate Harten-Lax-van Leer-Contact (HLLC) Riemann solver [36] is implemented for the inter-cell numerical flux. For the evaluation of the viscous fluxes the extrapolated interface variables $U_{i,a}^T$ and their unlimited gradients $\nabla U_{i,a}^T$ from the k-exact least-square reconstruction are averaged from two discontinuous states as detailed in [9,37]. For the gradients however additionally penalty terms are
included following the formulation of Gassner et al. [38] for suppressing odd-even decoupling modes in the numerical solutions [39], in the following manner:
\[ \nabla U = \frac{1}{2} (\nabla U_L + \nabla U_R) + \frac{\alpha}{L_{\text{int}}} (U_R - U_L) \hat{n}, \]
where \( L_{\text{int}} \) is the distance between the cell-centres of the adjacent cells, and \( \alpha = 4/3 \) similarly to [39,40].

2.2.5. Time stepping algorithms

Having constructed the numerical fluxes \( F_n \) as expressed in the semi-discrete conservative formulation, the next step involves the advancement of the solution in time. The explicit Strong Stability Preserving (SSP) Runge–Kutta 3rd-order method [41] has been employed for the time integration, and a CFL number of 0.9 for unsplit finite volume schemes [36] is used for all the test-cases (Fig. 4).

3. Code profiling and performance

The UCNS3D solver, is written in Fortran 2003 programming language, making use of object oriented programming, including abstract data types. It employs the message passing interface (MPI), and the Open Multi-Processing (OpenMP) application programming interface (API). The Metis partitioner [42] is used to decompose the mesh to numerous partitions; the total number of partitions is equal to the number of MPI processes. The work flow of the solver is composed into two main phases: the initialisation and the run-time as shown in Fig. 2.

Processes 1–7 correspond to the initialisation phase and the processes 8–15 correspond to the time advancement stage. In the
present study the main focus would be to improve the performance of the code for the run-time phase, therefore the I/O operations and the initialisation phases of the simulations are not accessed. As it can be seen in Fig. 2, the grey coloured boxes correspond to processes that are associated with floating point operations, the pink coloured boxes correspond to processes associated with communication between CPUs. In the following subsection details regarding the most expensive sections both in terms of computational and communication are discussed.

### 3.1. Least-squares reconstruction

For the least-squares reconstruction as shown in Eq. (16) the Moore–Penrose pseudo-inverse of $A_{mk}$ matrix, $A^\dagger_{mk}$ is stored in the computer memory at the initialisation phase and the polynomial coefficients are therefore obtained through a matrix vector multiplication of $A^\dagger_{mk}$ with $b_m$ as

$$a_k = A^\dagger_{mk} b_m. \tag{30}$$

This operation is repeated for all the conserved variables, and for all the admissible stencils. As the order of the polynomial increases the size of these matrices grows rapidly as shown in Table 1. This operation is implemented by making use of the intrinsic Fortran function `MATMUL` and is performed as shown in the Algorithm 1.

<table>
<thead>
<tr>
<th>Polynomial order $r$</th>
<th>Number of elements $m$ per stencil</th>
<th>Number of degrees of freedom $k$ per stencil</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>70</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>128</td>
<td>56</td>
</tr>
<tr>
<td>6</td>
<td>168</td>
<td>84</td>
</tr>
</tbody>
</table>

Algorithm 1 Pseudocode for least-square reconstruction procedure for cell $i$.

1. **procedure** LSQ_RECON($a_{k,j,s}$)
   2. **for** each admissible stencil $s$ **do**
      3. **for** each variable $j$ **do**
         4. $a_{k,j,s} = \text{MATMUL}(A^\dagger_{mk,s}, b_{m,j,s})$
      5. **end for**
   6. **end for**
7. **end procedure**

### 3.2. WENO weights computation

The weights of Eq. (21) are computed for all element’s face of the considered element, and for all the admissible stencils. As the order of the polynomial increases the size of these matrices grows fast as shown in Table 1. This operation is implemented by making use of the intrinsic Fortran function `MATMUL` and `DOTPRODUCT` and is performed as shown in the Algorithm 2.

Algorithm 2 Pseudocode for WENO weights computation for cell $i$.

1. **procedure** WENO_WEIGHTS($a_{k,j,s}$)
   2. **for** each face $f$ **do**
      3. **! Averaging**
         4. Compute $U_f^i = \frac{1}{2} (U_{L,f} + U_{R,f})$
         5. Compute eigenvectors $R_i$ and $L_i$
      6. **for** each admissible stencil $s$ **do**
         7. **for** each degree of freedom $k$ **do**
             8. **! Project to characteristic variables**
                9. Compute $B_{j,k}^s = \text{MATMUL}(L_{j,k}, a_{j,k}^s)$
             10. **end for**
         11. **end for**
      12. **end for**
   13. **end procedure**

3.3. Communication of stencil elements variables

The first communication requirement between different processors, is to exchange the mean variables of the elements in the stencils $b_{m,j,s}$ that belong to other processors, in order to solve the least-square reconstruction problem. It must be noted that due to the nature of the WENO directional stencils, and the geometrical sectors defined by the considered cells, there are instances where due to the high-aspect ratio cells the directional stencils might extend farther away from the central stencil. This implies that there might be cases where information might be needed from some processors, that they might not require any information from the current processor. Therefore resulting in cases where it is only needed to receive rather than send data from some processors. The `MPI_SENDRECV` function is used as shown in Algorithm 3.

### 3.3.1. Communication of stencil elements variables

3.4. Communication of variables/grads at quadrature points of halo cells

The second requirement between different processors, is to exchange the reconstructed variables and their gradients at each element’s interface between processor blocks for each Gaussian quadrature point. The operation is performed by using the `MPI_SENDRECV` function as shown in Algorithm 4.
Algorithm 3 Pseudocode for extrapolation of variables/gradients at cell faces $i$.

1: procedure EXTRAPOL($\mathbf{U}(x_0, t)$, $\nabla \mathbf{U}(x_0, t)$)
2: for each face $i$ do
3: for each Gaussian quadrature point $\alpha$ do
4: for each degree of freedom $k$ do
5: Compute the basis functions $\phi_k(\xi, \eta, \zeta)$
6: end for
7: ! Convective part
8: for each variable $j$ do
9: Compute the sum $\sum_{k=1}^{K} a_{j,k} \phi_k(\xi, \eta, \zeta)$
10: Compute the reconstructed solution as $U_{i,j,\alpha} = U_{i,j} + \sum_{k=1}^{K} a_{j,k} \phi_k(\xi, \eta, \zeta)$
11: end for
12: ! Diffusive part
13: Using only the central stencil $s = 1$
14: for each variable $j$ do
15: for each degree of freedom $k$ do
16: Obtain $\frac{\partial \phi_k(\xi, \eta, \zeta)}{\partial \xi}$
17: end for
18: Compute $\frac{\partial U_{i,j}^{\alpha}(\xi)}{\partial \xi} = \sum_{k=1}^{K} \frac{\partial \phi_k(\xi, \eta, \zeta)}{\partial \xi} a_{k,j}$
19: end for
20: ! Stencil
21: $\nabla U_{i,j,\alpha}(\mathbf{x}) = \mathbf{MATMUL}(J^{-1})_{i,j} \nabla U_{i,j,\alpha}(\xi)$
22: end for
23: end for
24: end procedure

Algorithm 4 Pseudocode for stencil elements communication.

1: procedure COMMSTENCIL($U_m$)
2: COMMUNICATION 1
3: for each CPU $J$ that only receives information from the current CPU $N$ do
4: Create 1D array to store the cell average values $U_m$ to be sent from CPU $N$
5: end for
6: for each CPU $J$ that only receives information from the current CPU $N$ do
7: CALL MPI_SENDRECV
8: end for
9: COMMUNICATION 2
10: for each CPU $L$ that only sends information to the current CPU $N$ do
11: Create 1D array to store the cell average values $U_m$ to be received from CPU $L$
12: end for
13: for each CPU $L$ that only sends information to the current CPU $N$ do
14: CALL MPI_SENDRECV
15: end for
16: COMMUNICATION 3
17: for each CPU $R$ that receives/sends information from/to the current CPU $N$ do
18: Create 1D array to store the cell average values $U_m$ to be received from CPU $R$
19: Create 1D array to store the cell average values $U_m$ of all the elements to be sent to CPU $R$
20: end for
21: for each CPU $R$ that receives/sends information from/to the current CPU $N$ do
22: CALL MPI_SENDRECV
23: end for
24: end procedure

Algorithm 5 Pseudocode for inter-processor quadrature points communications.

1: procedure COMMBOUND($\mathbf{U}(x_0, t)$, $\nabla \mathbf{U}(x_0, t)$)
2: for each CPU $P$ that receives/sends information from/to the current CPU $N$ do
3: Create 1D array to store the boundary extrapolated values and gradients to be received from CPU $P$
4: Create 1D array to store the boundary extrapolated values and gradients to be sent to CPU $P$
5: end for
6: for each CPU $P$ that receives/sends information from/to the current CPU $N$ do
7: CALL MPI_SENDRECV
8: end for
9: end procedure

4. Numerical setup

This section presents the results obtained for various three-dimensional test cases. The performance of the original implementation of CFD code is assessed in terms of scalability and efficiency as well as identifying the most time-consuming procedures. Then, a justification of the improvements that were pursued are detailed and finally the performance of the revised implementation of the CFD code is discussed. For all test cases the statistics in terms of maximum computational time, and maximum communication time are collected for 1000 iterations and the process is repeated at least three times to ensure reproducibility of the statistics and finally the average of those is used.

4.1. Data sets

For the numerical experiments, two test-problems are assessed of different computational and communication requirements. The first test problem is the ILES of the viscous Taylor–Green Vortex flow at a Reynolds number of $Re = 1.600$ which is used to study vortex stretching and dissipation characteristic of numerical schemes. It has been widely employed by many authors to quantify the dissipation behaviour of various high-order methods [43–48]. For the Taylor–Green Vortex flow case, two grids are utilised consisting of 0.2 and 2.1 million tetrahedral elements and two numerical schemes a WENO 2nd-order and a WENO 6th-order since they have vastly different computational requirements and hence will provide a representative indication of the relative performance improvements. Indicative results obtained on the fine mesh with a WENO 6th-order scheme are shown in Fig. 5 where the turbulent structures are shown with the q-criterion isosurfaces for two instances, one at the start and one at the end of the simulation.

The second test case is an ILES of the transitional turbulent subsonic flow past the SD7003 wing at an angle of attack of 8°, a Mach number of 0.2, and a Reynolds number of $Re = 60.000$. This test case is widely employed for assessing the behaviour of various numerical schemes for external aerodynamics by Visbal [49], Uranga et al. [50], Rizzetta and Visbal [51], Garmann et al. [52], Beck et al. [53], Bassi et al. [54], and Vermeire and Vincent [23],
is used with Intel MPI 5.1.3 and Intel Fortran compiler version 16.0.3. For all the runs apart from ARCHER (KNL) platform the exact compilation flags are set as -r8 -O3 -ipo -xHost -fp model precise. For the ARCHER (KNL) platform the MPI+OpenMP implementation is employed with Intel MPI 2017, and Intel Fortran compiler version 17.0.0. For all the runs in the ARCHER (KNL) platform the compilation flags are set as -i4 -r8 -O3 -xmic-avx512 -ipo -fp-model precise -gopenmp -gopenmp-link=static and a guided schedule is used for all the OpenMP regions, since it was found to be fastest. Enabling the inter-procedural optimisation has shown to have a significant effect on the performance, approximately 30% speed up and it is therefore used for all the tests. The computational times presented hereafter correspond to the maximum time across all CPUs, in other word the slowest CPU was taken into account for all the simulations, since it was found to be the most representative of the time that the simulation will take.

4.3. Original code performance

Assessing the performance of the original code implementation the Taylor Green Vortex on the finest tetrahedral mesh test case was utilised using the WENO 6th-order scheme. The purpose of this initial test is to identify the time-consuming processes of the simulation, and proceed with optimisation of only the most compute intensive operations. The test was performed on the Hazelhen HPC platform using 384, 3072 and 12,288 cores. The breakdown of the percentage of the total time taken by various processes is illustrated in Fig. 7 top three pie charts. It can be noticed that the WENO weights calculation is the most expensive operation followed by the least-square reconstruction and the extrapolation of the flow variables and gradients at the cell interfaces. In terms of communication cost the most expensive part is the communication of the reconstructed solutions and their gradients for the Gaussian quadrature points between inter-processor boundaries rather than the communication of the solutions for the stencil elements across inter-processor boundaries. Therefore only three processes were identified as having the potential to significantly improve the performance of the CFD code namely the WENO weights calculation, the least-squares reconstruction and the communication for the Gaussian quadrature points between inter-processor boundaries and the optimisation of those will be pursued.
Table 2
Description of HPC resources used for the numerical tests.

<table>
<thead>
<tr>
<th>Specifications</th>
<th>Hazelhen (HW)</th>
<th>SuperMUC (SB)</th>
<th>SuperMUC (HW)</th>
<th>ARCHER (IB)</th>
<th>ARCHER (KNL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computing Site</td>
<td>High Performance Computing Center Stuttgart (HRLS)</td>
<td>Leibniz Supercomputing Centre (LRZ)</td>
<td>Leibniz Supercomputing Centre (LRZ)</td>
<td>UK National Supercomputing Centre (EPCC)</td>
<td>UK National Supercomputing Centre (EPCC)</td>
</tr>
<tr>
<td>Location</td>
<td>Stuttgart, Germany</td>
<td>Garching, Germany</td>
<td>Garching, Germany</td>
<td>Edinburgh, UK</td>
<td>Edinburgh, UK</td>
</tr>
<tr>
<td>Processor Type</td>
<td>Intel Haswell E5-2680 v3</td>
<td>Intel Sandy Bridge E5-2680</td>
<td>Intel Haswell E5-2697 v3</td>
<td>Intel Ivy Bridge E5-2697 v2</td>
<td>Intel Knights Landing Xeon Phi (7210)</td>
</tr>
<tr>
<td>Processor Frequency</td>
<td>2.5 GHz</td>
<td>2.7 GHz</td>
<td>2.6 GHz</td>
<td>2.7 GHz</td>
<td>1.3 GHz</td>
</tr>
<tr>
<td>Cores per node</td>
<td>24</td>
<td>16</td>
<td>28</td>
<td>24</td>
<td>64 (4 hardware threads)</td>
</tr>
<tr>
<td>Memory per node</td>
<td>128 GB</td>
<td>32 GB</td>
<td>64 GB</td>
<td>64/128 GB</td>
<td>96 GB and 16 GB on chip (MCDRAM)</td>
</tr>
<tr>
<td>Number of nodes</td>
<td>7712</td>
<td>9216</td>
<td>3072</td>
<td>4544 (64 GB), 376 (128 GB)</td>
<td>12</td>
</tr>
<tr>
<td>Interconnect</td>
<td>Cray Aries</td>
<td>Infiniband FDR10</td>
<td>Infiniband FDR14</td>
<td>Cray Aries</td>
<td>Cray Aries</td>
</tr>
</tbody>
</table>

Fig. 7. Top: baseline code performance profile, bottom: optimised code performance profile. 3D Taylor Green Vortex test problem using a WENO 6th-order scheme on the finest tetrahedral mesh, at Hazelhen HPC platform.

4.4. Reconstruction optimisation performance

The two processes associated with the reconstruction method including the WENO weights computation and the least-square reconstruction are further assessed using the Intel VTune profiler. It was confirmed what already has been seen from manual time measurement and log output, the main computational hotspot is the WENO weights calculation algorithm. Within this routine the most costly part was the projection to characteristic variables for each degree of freedom and the weights calculation, since this is a process repeated for all the stencils, and all the faces of the considered cell. For both processes most of the time was spent in dense linear algebra operations including matrix-vector products and dot products using the intrinsic Fortran MATMUL functions. The key
ingredients to improve the performance of these routines are listed below:

1. Modify the storage of multidimensional arrays to optimise memory access patterns.
2. Rewrite loops of matrix-vector multiplications, into matrix-matrix multiplications when possible.

4.4.1. Least-square reconstruction optimisation

For the least-square reconstruction, outlined in Algorithm 1, we improve the memory access patterns by storing all matrices with the fastest varying index stored first following the Fortran column major order. Moreover, the loops are organised in such a way that the largest index is looped last in order to minimize loop overheads. The matrix-vector product of MATMUL in line 4 of Algorithm 1 is replaced with the equivalent call to GEMV provided by Fortran 95 BLAS binding from Intel MKL version 11.3. This can be seen in Table 3, in the columns “Original Code” and “Preliminary Code (GEMV)”, in that way the computation of the least-square reconstruction is speeded up by a factor of 2. Pursuing additional optimisation within the loop where all the conserved variables are computed in line 3 of Algorithm 1, by removing the MATMUL Fortran function to perform the matrix-vector multiplication of the pseudo inverse matrix and the vector of each of the conserved variables for all the stencil elements. This function is replaced with a matrix-matrix multiplication of the pseudo-inverse matrix with the matrix containing the conserved variables b_{m,j,s} for all the stencil elements using the GEMM library function as shown in Algorithm 6. This modification resulted in the least-square reconstruction process to take 5 times less time as shown in Table 3.

4.4.2. WENO weights computation optimisation

For the WENO weights computation a similar optimisation strategy as the least-square reconstruction is adopted. MATMUL functions are replaced by calls to GEMV BLAS library functions. This optimisation resulted in a speed up of a factor of 5 compared to the original code as it can be seen in Table 3. By eliminating the loops performed at lines 7, 11 and 31 of Algorithm 2, and replacing the MATMUL function of the matrix-vector product with the GEMM BLAS library functions of the equivalent matrix-matrix products, and the DOTPRODUCT by the DOT BLAS library functions as seen in Algorithm 7 a speed-up of 10 is achieved compared to the original demonstrated in Table 3.

Table 3

<table>
<thead>
<tr>
<th>Routine</th>
<th>Original code time</th>
<th>Preliminary code (GEMV) Time</th>
<th>Final code (GEMM) Time</th>
<th>Total speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>WENO Reconstruction</td>
<td>3073 s</td>
<td>1466 s</td>
<td>478 s</td>
<td>×</td>
</tr>
<tr>
<td>WENO weights</td>
<td>2135 s</td>
<td>429 s</td>
<td>197 s</td>
<td>×</td>
</tr>
<tr>
<td>LSQ Reconstruction</td>
<td>780 s</td>
<td>367 s</td>
<td>157 s</td>
<td>×</td>
</tr>
</tbody>
</table>

Algorithm 6 Pseudocode of optimised Least-square reconstruction procedure for cell i.

1: procedure LSQ RECON(a_{k,j,s})
2: for each admissible stencil s do
3: \( a_{k,j,s} = \text{GEMM}(A_{mk,s}^T, b_{m,j,s}) \)
4: end for
5: end procedure

Algorithm 7 Pseudocode of optimised WENO weights computation for cell i.

1: procedure WENO WEIGHTS(a_{k,j,s})
2: for each face l do
3: \( ! \text{Averaging} \)
4: Compute \( U_n = \frac{1}{2}(U_{l,n} + U_{k,n}) \)
5: Compute eigenvectors \( \mathbf{R}_l \) and \( \mathbf{L}_l \)
6: for each admissible stencil s do
7: \( ! \text{Project to characteristic variables} \)
8: Compute \( B^s_{j,ikl} = \text{GEMM}((L_j, \mathbf{a}_{k,j,s}) \)
9: \( S_k = \text{GEMM}(C_{T_kq}, B^s_{j,ik}) \)
10: for each conserved variable j do
11: \( S_j = \text{DOT}(S_k, B^s_{j,ik}) \)
12: end for
13: end for
14: \( ! \text{Compute Weights} \)
15: for each conserved variable j do
16: Set \( \tilde{a}_0 = 0, a_0 = 0 \)
17: for each admissible stencil s do
18: Compute \( \tilde{a}_l = \frac{\lambda_l}{\sigma^2 + \tilde{a}_l^2} \)
19: end for
20: Compute sum \( \sum_{j=1}^{n_l} \tilde{a}_l \)
21: for each admissible stencil s do
22: Compute \( a^l = \frac{\tilde{a}_l}{\sum_{j=1}^{n_l} \tilde{a}_l} \)
23: end for
24: end for
25: for each degree of freedom k do
26: \( \tilde{B}_{j,ik} = \left( \sum_{s=0}^{n_l} \tilde{a}_l B^s_{j,ik} \right) \)
27: end for
28: \( ! \text{Project to conserved variables} \)
29: Compute \( \tilde{a}_{j,ik} = \text{GEMM}(\mathbf{R}_{j,l}, \tilde{B}_{j,ik}) \)
30: end for
31: end procedure

4.5. Communication optimisation performance

The Communication of stencil elements variables has a much smaller footprint in terms of time taken compared to the communication of the boundary extrapolated variables and gradients for all the test runs performed. Therefore the optimisation efforts were focused on the communication of the boundary extrapolated variables and gradients. The MPI-3 sparse collective operations is considered for the optimisation of the boundary extrapolated variables and gradients communication exchange at each Gaussian quadrature point at each element/block interface. The collective operations with MPI Types such as MPI_Dist_graph_create_adjacent and MPI_Neighbour_alltoall for avoiding the overhead of manual send and receive buffer packing and unpacking are considered. However, it got apparent that the time required for copying data to and from send and receive buffers was negligible.
compared to the communication of the messages and therefore this approach was not implemented. On the other hand the blocking MPI_SENDRECV calls are replaced by non-blocking communications of MPI_ISEND, MPI_IRECV and MPI_WAITALL as shown in Algorithm 8. This resulted in an improvement of 16% with respect to the original code as shown in Table 4.

4.6. Final performance gains

The performance gains of the revised code are assessed on the same tests and the same number of processors for the Taylor Green Vortex on the finest tetrahedral mesh test case using a WENO 6th-order scheme. The purpose of this test is to identify significant differences in the breakdown of computational time similarly to the Fig. 7. The tests are performed on the Hazelhen HPC platform using 384, 3072 and 12,288 cores. The breakdown of the percentage of the total time taken by processes is illustrated in Fig. 7. WENO weights process have equivalent cost to the least-square reconstruction, also noticed in Table 3. The percentage of total time taken by the communication of the boundary extrapolated values and gradients is reduced from 24% to 20%.

Since the computational load increases as the polynomial order gets higher, we are expecting different gains for different orders of accuracy for the same test problem. This is reflected by the performance improvements of the greater size of the matrix operations while employing the GEMM library calls. At the same time there are overheads associated with making calls to the external library. However, these overheads become increasingly insignificant with higher order discretisation.

We perform simulations employing the WENO 2nd-order scheme on the finest tetrahedral mesh for the Taylor-Green vortex on the SuperMUC (SB) cluster by performing a strong scalability test of the original code and the optimised code. The simulations are performed from 48 to 1024 CPUs as seen in Fig. 8: there isn’t significant drop in performance in terms of scalability. It apparent that optimised code is approximately 50% faster per iteration than the original code, even with the overheads associated with GEMM calls, since the cumulative improvements. It has to be noted that on 1024 CPUs, approximately 2050 elements are assigned to each CPU.

The second test is composed of simulation employing the WENO 6th-order scheme on the finest tetrahedral mesh for the Taylor-Green vortex, on the SuperMUC (SB) and (HW) cluster, and on the Hazelhen (HW) cluster. The goal is to perform strong scalability test of both codes and assess the improvement across different machines. The simulations are performed from 384 to 12,288 CPUs as seen in Fig. 9. The speed up for the WENO 6th-order scheme spans from 5 to 6.5, across all CPU counts. Similar performance is obtained for the simulation running on 384 CPUs on the revised code with the same case running on 2400 CPUs on original code. It needs to be stressed that at the highest core counts each CPU holds is approximately 160 elements, highlighting that the ratio of computational to communication load is still high even for low number of elements assigned to each CPU block. Furthermore, the scalability performance hold at a acceptable level for the number of cpu tested, suggesting that higher core counts could be explored with lower number of elements per CPU. For very high-order of accuracy it is expected that the improvements in performance will be even greater, since the performance of the GEMM library calls are proportional to the size of the matrices.

The flow past the SD7003 wing is considered in the context of ILES; the discretisation scheme employed is the WENO 4th-order scheme and the case is run on the SuperMUC (HW) cluster. The simulations are performed from 864 to 7200 CPUs as seen in Fig. 10. Speed up for the WENO 4th-order scheme ranges between 1.8 and 2.2, across all CPU counts. The WENO 4th-order scheme
is not experiencing the same performance increase as the WENO 6th-order scheme. This can be attributed to the fact that the polynomial order is smaller and the fact that mixed-element grids introduces additional overheads in terms of element imbalance between partitioned blocks.

We assess the performance and scalability of the optimised solver on the ARCHER Knights Landing-(KNL) platform cluster composed of Xeon Phi processors. The test case is the Taylor-Green vortex flow with the WENO 6th-order on a coarse resolution grid composed of 0.2 million elements. The performance gains of using the GEMM library calls in an architecture with a much wider 512-bit vector units, and faster memory bandwidth.

The simulations are performed on the (KNL) from 1 to 8 nodes as seen in Fig. 11. A speed up from 7 to 8.5 is achieved across all CPU counts with the optimised code compared with the original one. The performance increase in this architecture is even more pronounced since the data for the simulation fits within the fast memory of the (KNL) processor. Due to the nature of the unstructured CFD solver, where non-coalescent memory access patterns are present, the increased memory bandwidth in combination with the much wider 512-bit vector units in this architecture results in significant performance improvements. Initially 64 MPI processes per Knights Landing chip were used, but this was found not to be the optimum configuration. With the utilisation of hybrid MPI+OPENMP implementation of the code, it was found that
the ideal combination for running the simulations was with 4 MPI processes per Knights Landing chip, and 64 threads spawned from each one of them, resulting a total of 256 threads per chip. By using the hybrid MPI+OpenMP a speed up of approximately 1.4 was observed compared to the pure MPI implementation of the revised code. Finally the utilisation of the ARCHER (KNL) platform results in higher computational performance per purchasing cost of the CPUs in each computational node, as compared to the ARCHER (IB) nodes as seen in Fig. 11. This is a good indication on the potential of these architectures for pushing the boundaries of efficiency and cost, for large scale scientific computing applications.

5. Conclusions

In this work the performance of high-order WENO schemes of the UCNS3D CFD solver is assessed under the ILES framework. The code is profiled, optimised and tested across five different HPC clusters. Optimisation of the most compute-hungry processes is performed i.e. WENO weights computation, least-square reconstruction and communication of inter-processor boundary extrapolated variables and gradients. Significant performance benefits are achieved by making use of BLAS library functions and replacing matrix-vector operations with matrix-matrix. Blocking MPI communication functions are replaced with non-blocking ones for the exchange of inter-block variables. Across five different HPC platforms the speed-up observed between the revised and the original code, spanned from 1.5 to 8.5. Finally, the use of hybrid MPI+OpenMP capability of the code resulted in an additional performance gains of 1.4 on the ARCHER Knights Landing cluster. The results obtained are encouraging as one of the major challenges of high-order schemes is the inherited cost. Future-proofing high-order solvers by incorporating enabling technologies for new and future hardware architectures will ensure that the high-order movement will remain relevant and better bridge the gap with the CFD workhorse of lower-order methods.

Acknowledgements

The authors would like to acknowledge the computing time at HAZELHEN at the High-Performance Computing Center Stuttgart (HLRS), Germany and SuperMUC at the Leibniz Supercomputing Centre (LRZ) in Garching, Germany in the framework of the PRACE project funded in part by the EU Horizon 2020 research and innovation programme (2014–2020) under grant agreement 653838. The authors would also like to acknowledge the computing on the UK national high-performance computing service ARCHER, that was provided through the UK Turbulence Consortium in the framework of the EPSRC grant EP/L000261/1.

References


