A Comparative Study on the Parallel Efficiency of Low- and High-Order Finite-Volume Schemes

Leonardo da Silva Rosa e Oliveira
leonardo.oliveira@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Portugal

December 2022

Abstract

Although the use of second-order numerical schemes remains widespread due to their convergence robustness and implementation ease, high-order schemes have emerged as promising alternatives due to their increased numerical accuracy and computational efficiency. However, one of the recurring arguments against the adoption of high-order schemes is that, even though they are more efficient than second-order schemes when executed serially, they perform poorly in parallel computing. The present work compared the parallel performance of second-, fourth-, sixth-, and eighth-order face least-squares (FLS) schemes, which had been previously developed for two-dimensional, unstructured grids under the finite-volume framework. The FLS algorithm was extended to three dimensions, and a corresponding code with parallel computing capabilities was developed. The parallel performance of the second- through eighth-order schemes was studied for a three-dimensional convection-diffusion problem using regular Cartesian grids, at the Oblivion supercomputer. The obtained results showed good scalability across all schemes, with the eighth-order scheme performing equivalently to the second-order one. Their memory and runtime efficiency were also analyzed for parallel execution, with the high-order schemes presenting an overwhelming advantage.

Keywords: High-order schemes, Parallel scalability, High-performance computing, Finite volume method, Convection-diffusion equation

1. Introduction

The development of fluid mechanics has historically followed a two-pronged approach based on theoretical derivation and experimental observation [1]. The development of computers in the twentieth century, however, birthed a "third approach" – computational fluid dynamics (CFD). The development of CFD accompanied and was aided by the growth in computing power over the twentieth century. The more recent shift to multi-core processors and massively parallel clusters in high-performance computing (HPC) has led to the corresponding need for methods and algorithms to take advantage of such resources. The National Aeronautics and Space Administration (NASA), in a 2014 report on the vision for state-of-the-art CFD in 2030, identified the "effective utilization of HPC" as one of five key strategic areas, with "robust CFD code scalability" listed as a major technology gap [2]. The authors pointed out that, at the time, most codes ran efficiently on at most ~1000 cores, which represented only ~0.1% capacity for the then-largest supercomputers.

Currently, most commercial solvers continue to use second-order accurate numerical schemes, because of their robustness in obtaining a converged solution and ease of implementation on any grid type. However, these schemes require very fine grid resolutions for accurate results, which may lead to prohibitively large computational problems in terms of memory requirement. A promising alternative arises in high-order schemes (usually defined as third-order and higher), since these can achieve the same error tolerance by using a coarser grid. This benefit has been shown for several grid types, including structured [3], curvilinear or deformed [4], and unstructured [5]. Additionally, Ekaterinaris [6] shows, through a simple rationale, that the use of high-order methods is also advantageous in terms of solver runtime, barring a drastic increase of the operation count for the linear solver.

High-order finite-volume (FV) methods have as one of its precursors the 1990 work by Barth and Frederickson [7], with quadratic reconstructions (third-order) on unstructured grids for the Euler equations. Since then, efforts led to the implemen-

tation of fourth-order schemes by Ollivier-Gooch et al. [8, 9, 10] and sixth-order schemes by Clain et al. [11, 12, 13]. For an extensive background on high-order FV schemes, refer to the comprehensive review articles on structured grids by Ekaterinaris [6] and on unstructured grids by Wang [14].

One of the general criticisms towards high-order FV schemes is that they lack robustness for flow regions of high gradient or discontinuities, being susceptible to spurious oscillations and thus possibly failing to converge [6]. There have been several methods developed to deal with these oscillations while preserving high-order accuracy, with a major family of these being the essentially non-oscillatory (ENO) schemes [15] and its variants – weighted ENO (WENO) [16, 17], central ENO (CENO) [18], and targeted ENO (TENO) [19]. Cueto-Felgueroso et al. [20, 21] provided an alternative by implementing the moving least-squares technique to FV schemes. This technique provides "shape functions" to the grid by means of least-squares reconstructions, which uses a weight function related to the distance between the reference face and the cell sample.

2. The Numerical Method

Consider the convection-diffusion equation – a general conservation equation present in fields such as fluid flow and heat transfer [1] – given by

$$\underbrace{\nabla\cdot(\vec{v}\varphi)}_{\text{convection}} - \underbrace{\nabla\cdot(\Gamma\nabla\varphi)}_{\text{diffusion}} = \underbrace{Q}_{\text{source}} \,. \tag{1}$$

Following the discretization procedure according to the FV framework [22], the first step in this method is to integrate (1) over a control volume (CV):

$$\int_{CV}\Big(\nabla\cdot(\vec{v}\varphi) - \nabla\cdot(\Gamma\nabla\varphi)\Big)\,dV = \int_{CV} Q\,dV \,. \tag{2}$$

By applying the Divergence Theorem on the previous equation, one is able to transform the volume integrals of the convective and diffusive terms into surface integrals over the surface of the CV – i.e. the control surface CS – resulting in

$$\int_{CS}\Big(\vec{v}\varphi - \Gamma\nabla\varphi\Big)\cdot d\vec{S} = \int_{CV} Q\,dV \,, \tag{3}$$

where the term $d\vec{S}$ represents the area normal of the surface, defined as positive outwards. Defining $\mathcal{F}(I)$ as the set of all faces of cell I, one then obtains

$$\sum_{f\in\mathcal{F}(I)}\int\Big(\vec{v}\varphi - \Gamma\nabla\varphi\Big)\cdot d\vec{S}_f = \int_{CV} Q\,dV \,. \tag{4}$$

The source term integration may be calculated using cubature rules for numerical integration:

$$\int_{CV} Q\,dV \simeq V\sum_{g\in\mathcal{G}(I)} w_{G_g} Q(\vec{x}_g)\,, \tag{5}$$

where $\mathcal{G}(I)$ is the set of cubature points associated with cell I, $\vec{x}_g$ is a vector representing the spatial position of cubature point g, and w_{G_g} is the weight of corresponding to the cubature point g according to the chosen cubature rule. Similarly, the surface integrals of the convective and diffusive terms in (4) can also be calculated using cubature rules,

$$\int\Big(\vec{v}\varphi - \Gamma\nabla\varphi\Big)\cdot d\vec{S}_f \simeq \sum_{g\in\mathcal{G}(f)} w_{G_g}\Big(\vec{v}\varphi(\vec{x}_g) - \Gamma\nabla\varphi(\vec{x}_g)\Big)\cdot\vec{S}_f \,, \tag{6}$$

where $\mathcal{G}(f)$ is the set of cubature points associated with face f. As such, combining (5) and (6) into (4) yields the semi-discretized convection-diffusion equation

$$\sum_{f\in\mathcal{F}(I)}\sum_{g\in\mathcal{G}(f)} w_{G_g}\Big(\vec{v}\varphi(\vec{x}_g) - \Gamma\nabla\varphi(\vec{x}_g)\Big)\cdot\vec{S}_f = V\sum_{g\in\mathcal{G}} w_{G_g} Q(\vec{x}_g)\,. \tag{7}$$

For the face least-squares (FLS) schemes used in the current work, the terms $\varphi(\vec{x}_g)$ and $\nabla\varphi(\vec{x}_g)$ – evaluated at the cubature point g – are approximated using a polynomial centered on face f. Let this regression polynomial be of the following form:

$$\begin{aligned}\varphi(\vec{x}_g) = \kappa_0 &+ \kappa_x(x_g - x_f) + \kappa_y(y_g - y_f) + \kappa_z(z_g - z_f) \\ &+ \kappa_{xx}(x_g - x_f)^2 + \kappa_{yy}(y_g - y_f)^2 + \kappa_{zz}(z_g - z_f)^2 \\ &+ \kappa_{xy}(x_g - x_f)(y_g - y_f) + \kappa_{xz}(x_g - x_f)(z_g - z_f) \\ &+ \kappa_{yz}(y_g - y_f)(z_g - z_f) + \dots\end{aligned} \tag{8}$$

In vector form, one may represent this approximation by the product between a coefficient vector $\boldsymbol{\kappa}_f$ and a vector $\mathbf{d}_f(\vec{x}_g)$ of the terms representing the distance between $\vec{x}_g$ and $\vec{x}_f$,

$$\varphi(\vec{x}_g) = \mathbf{d}_f(\vec{x}_g)\,\boldsymbol{\kappa}_f \,. \tag{9}$$

However, this still leaves the coefficient vector $\boldsymbol{\kappa}_f$ as unknown. Now, consider a neighboring cell near the same face f. Using (9), the value of φ at this cell's centroid $\vec{x}_{\text{nb}}$ may be written as

$$\varphi(\vec{x}_{\text{nb}}) = \mathbf{d}_f(\vec{x}_{\text{nb}})\,\boldsymbol{\kappa}_f \,. \tag{10}$$

Therefore, grouping several these neighboring cells into a stencil $s(f)$ and combining their individual expressions for φ yields the following matrix-vector equation

$$\boldsymbol{\varphi}_{s(f)} = \mathbf{D}_{s(f)}\,\boldsymbol{\kappa}_f \,. \tag{11}$$

where $\boldsymbol{\varphi}_{s(f)}$ contains φ_c, $c\in s(f)$ and $\mathbf{D}_{s(f)}$ is a matrix created by concatenating vectors $\mathbf{d}_f(\vec{x}_c)$, $c\in s(f)$. In an expanded matrix form, the previous equation is given by (12).

$$\begin{bmatrix} \varphi_{c_1} \\ \varphi_{c_2} \\ \vdots \\ \varphi_{c_N} \end{bmatrix} = \begin{bmatrix} 1 & x_{c_1}-x_f & y_{c_1}-y_f & z_{c_1}-z_f & (x_{c_1}-x_f)^2 & (x_{c_1}-x_f)(y_{c_1}-y_f) & \cdots \\ 1 & x_{c_2}-x_f & y_{c_2}-y_f & z_{c_2}-z_f & (x_{c_2}-x_f)^2 & (x_{c_2}-x_f)(y_{c_2}-y_f) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \\ 1 & x_{c_N}-x_f & y_{c_N}-y_f & z_{c_N}-z_f & (x_{c_N}-x_f)^2 & (x_{c_N}-x_f)(y_{c_N}-y_f) & \cdots \end{bmatrix} \begin{bmatrix} \kappa_0 \\ \kappa_x \\ \kappa_y \\ \vdots \end{bmatrix} \tag{12}$$

This allows one to pose the issue of determining $\mathbf{c}_f$ as a weighted least-squares problem expressed as the minimization of a residual function, as given by

$$r\left(\boldsymbol{\varphi}'_{s(f)}\right) = \left(\boldsymbol{\varphi}_{s(f)} - \boldsymbol{\varphi}'_{s(f)}\right)^{\mathrm{T}} \mathbf{W}_{s(f)} \left(\boldsymbol{\varphi}_{s(f)} - \boldsymbol{\varphi}'_{s(f)}\right) . \tag{13}$$

In this expression, $\boldsymbol{\varphi}$ and $\boldsymbol{\varphi}'$ are, respectively, the original data set of the generated stencil $s(f)$ and the corresponding obtained solution. Furthermore, $\mathbf{W}_{s(f)}$ is a diagonal matrix containing the weights assigned to each cell in the stencil. The inclusion of weights gives a greater accuracy to the method by allowing cells closer to the target face f to be given a greater influence on the value of $\boldsymbol{\kappa}_f$. The current work will only consider the face-neighboring stencil algorithm developed by Diogo [23] and use the recommended value of $w_c = |\vec{x}_c - \vec{x}_f|^{-6}$, $c \in s(f)$.

Multiplying both sides of (11) by $\mathbf{D}_f^T \mathbf{W}_{s(f)}$ and then performing some algebraic manipulation, one obtains an equation for the coefficient vector $\boldsymbol{\kappa}_f$

$$\boldsymbol{\kappa}_f = \mathbf{P}_{s(f)} \boldsymbol{\varphi}_{s(f)} , \tag{14}$$

$$\mathbf{P}_{s(f)} \equiv \left(\mathbf{D}_{s(f)}^T \mathbf{W}_{s(f)} \mathbf{D}_{s(f)}\right)^{-1} \mathbf{D}_{s(f)}^T \mathbf{W}_{s(f)} \tag{15}$$

where $\mathbf{P}_{s(f)}$ is the pseudo-inverse matrix.

Then, substituting the obtained expression for $\boldsymbol{\kappa}_f$ into (9) allows for $\varphi(\vec{x}_g)$ to be calculated from the corresponding stencil values, as shown in (16). Similarly, $\nabla\varphi(\vec{x}_g)$ can be calculated from (17). In the particular case of regular Cartesian grids, the values in coefficient vector $\mathbf{d}_f(\vec{x}_g)$ depend only on cubature point g, since the distribution of cubature points will be the same for any face; as such, the subscript f has been omitted. Additionally, many of these terms in $\mathbf{d}_g$ will be null; therefore, only a small portion of the matrix multiplication $\mathbf{d}_g \mathbf{P}_f$ needs to be computed. Unfortunately, the computation of $\mathbf{P}_{s(f)}$ itself is expensive, due to the need to compute an inverse matrix.

$$\varphi(\vec{x}_g) = \mathbf{d}_f(\vec{x}_g) \mathbf{P}_{s(f)} \boldsymbol{\varphi}_{s(f)} \tag{16}$$

$$\nabla\varphi(\vec{x}_g) = \nabla\mathbf{d}_f(\vec{x}_g) \mathbf{P}_{s(f)} \boldsymbol{\varphi}_{s(f)} \tag{17}$$

Substituting these two expressions into (7) leads to the final equation (18) used in the method, which will be used to create the linear system $\mathbf{A}\boldsymbol{\varphi}_{s(f)} = \mathbf{b}$.

$$\sum_{f\in\mathcal{F}(I)} \sum_{g\in\mathcal{G}(f)} w_{G_g} \Big[\left(\vec{S}_f \cdot \vec{v}\right) \mathbf{d}_f(\vec{x}_g) \mathbf{P}_{s(f)} - \Gamma \left(\vec{S}_f \cdot \nabla\right) \mathbf{d}_f(\vec{x}_g) \mathbf{P}_{s(f)} \Big] \boldsymbol{\varphi}_{s(f)} = V \sum_{g\in\mathcal{G}(I)} w_{G_g} Q(\vec{x}_g) \tag{18}$$

3. Verification of Numerical Schemes

To verify the implementation of the FLS schemes, the author performed a series of tests to measure each method's accuracy and convergence order. The analytical functions used are given by

$$\varphi_{2\mathrm{D}}(\vec{x}/3\pi) = \sin x \sin y , \tag{19}$$

$$\varphi_{3\mathrm{D}}(\vec{x}/3\pi) = \sin x \sin y \sin z . \tag{20}$$

These functions were chosen because they allow for the simultaneous verification of several terms, since they have both zero and non-zero boundary values and produce a non-zero source term when inserted into the convection-diffusion equation.

Since the purpose of these tests was to verify the novel implementation in C using PETSc – and not the FLS method itself – the only boundary conditions considered were the Dirichlet type. Note that Diogo [23] verified that the FLS method also works correctly for Neumann- and Robin-type boundary conditions. The error norms and convergence orders were calculated for all FLS schemes in several grid sizes and also for a second-order central differences (CD2) scheme as a classical reference. Henceforth, the notation used to identify an n-order FLS method will be FLSn.

The error norms $\|\boldsymbol{\varepsilon}\|_1$ for each case are plotted as functions of the cell's reference length $L_{\mathrm{ref}} \equiv (N_C)^{1/N_D}$, where N_C is the number of cells in the mesh and N_D is the number of spatial dimensions.

3.1. Two-Dimensional Case

In order to obtain more relevant value for the finer grid sizes, the test was run in 128-bit quadruple precision. This allowed for a delayed appearance of round-off errors, albeit with a much higher computational cost; such decision presented an acceptable trade-off since the test's objective was to confirm numerical accuracy and not computational performance. The resulting system of equations was solved with the BiCGSTAB solver preconditioned using block-Jacobi. The evolutions of the

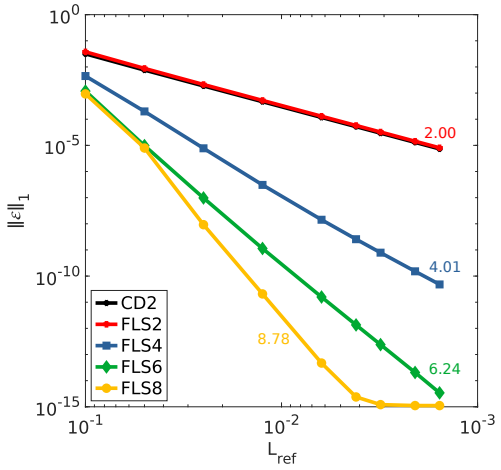

Figure 1. Error norm $\|\boldsymbol{\varepsilon}\|_1$ for 2D FLS schemes and CD2, using the analytical solution (19).

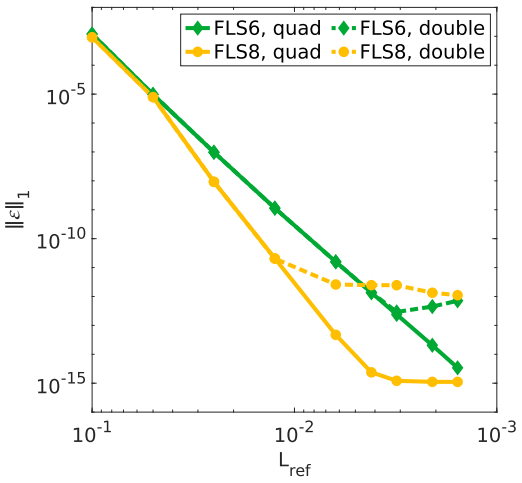

Figure 2. Influence of floating-point precision – quadruple (solid) or double (dashed) – on $\|\boldsymbol{\varepsilon}\|_1$.

error norm with respect to the reference length L_{ref} for the two-dimensional case is plotted in Figure 1.

In terms of convergence order, the results clearly show that all FLS schemes exhibit their expected asymptotic behavior. The only small exception to this is FLS8 for error norms of $\lesssim$1E-15, when the round-off error starts to dominate the global numerical error and the scheme then loses its theoretical convergence order. In terms of numerical accuracy, the obtained results agree with those obtained by Diogo [23], save for fluctuations due to a different choice of solver. Note also that the FLS2 scheme presents a slightly larger error than its CD2 counterpart, since the latter's numerical error contains additional contributions from the weighted least-squares regression error. Overall, these results demonstrate the correct implementation of the two-dimensional FLS schemes.

3.2. Quadruple- vs. Double-Precision Results

The results shown above were obtained using quadruple precision, which helps delay the contamination from round-off errors [24]. Nonetheless, in Figure 1 this limit is clearly identifiable by the FLS8 plateau around $\|\boldsymbol{\varepsilon}\| \sim 10^{-15}$. If one were to use double precision instead, this value would be even higher. This is shown in Figure 2, which plots the results obtained using quadruple and double precision for chosen FLS schemes alongside each other. The use of quadruple precision leads to a significant improvement in the numerical accuracy for finer meshes with $N_C \gtrsim 100^2$, which quickly become dominated by round-off errors when using double precision. Notice that the saturation values when using double precision are consistent with the results obtained by Diogo [23] for both the FLS6 and FLS8 schemes.

3.3. Three-Dimensional Case

As with the previous case, the test was run in 128-bit quadruple precision using BiCGSTAB solver preconditioned with block-Jacobi. The error norm $\|\boldsymbol{\varepsilon}\|_1$ for the three-dimensional case is plotted in Figure 3, with the corresponding convergence orders annotated for each method.

Similarly to the 2D case, the 3D implementation of all FLS schemes exhibit the expected asymptotic behavior. Unlike the previous case, the results do not enter a region where round-off error dominates. This happens because the global error is not small enough and the 3D problem is memory-limited – i.e. using a finer grid would exceed the system's available memory capacity.

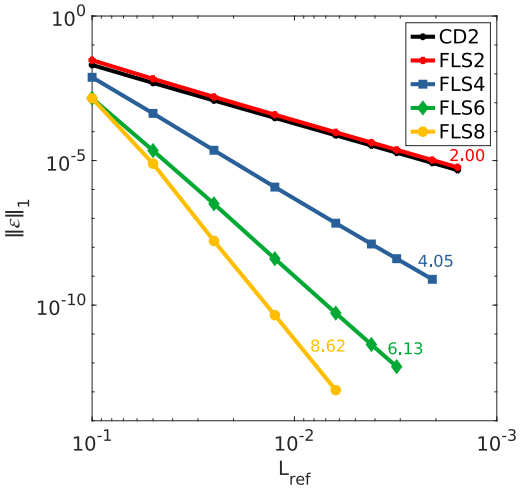

Figure 3. Error norm $\|\boldsymbol{\varepsilon}\|_1$ for 3D FLS schemes and CD2, using the analytical solution (20).

Table 1. Preconditioner-solver combination for FLS2, serial runtime in seconds.

KSP \ PC	BJACOBI	ASM				GAMG			
		0	1	2	3	0	2	4	6
CG	4412	2625	2530	3279	2600	11 095	133	82	93
BICG	716	834	832	832	831	-	-	-	-
BCGS	548	1063	737	733	739	71	66	73	97
IBCGS	451	563	561	555	567	-	-	-	-
GMRES	2778	4521	2861	3048	2984	61	58	66	85
FGMRES	6076	4990	3700	3639	3037	62	73	82	95
TFQMR	5263	5571	4565	4852	4384	58	54	63	77

Table 2. Preconditioner-solver combination for FLS2, parallel runtime in seconds.

KSP \ PC	BJACOBI	ASM				GAMG			
		0	1	2	3	0	2	4	6
CG	277	341	-	-	-	1700	13	-	-
BICG	138	191	21	14	7	-	-	-	-
BCGS	100	125	129	113	100	12	11	-	-
IBCGS	110	133	89	110	121	-	-	-	-
GMRES	-	-	414	414	419	11	10	-	-
FGMRES	486	683	719	625	594	13	14	-	-
TFQMR	107	140	-	-	-	12	11	-	-

4. Selection of Preconditioner and Solver

This section describes the preliminary studies done with the two-dimensional code to help choose a preconditioner-solver combination for the full-fledged studies on parallel performance (see Section 5). Both versions of the code use the same approach to solve the linear system resulting from the discretization process. Using the PETSc framework, the system is solved with an iterative linear solver, through the combination of a preconditioner (PC) and a Krylov subspace iterative method (KSP) [25]. Table 3 lists the full names and PETSc acronyms of the preconditioners and solvers considered in this work.

The main goal of this preliminary study was to reduce the myriad choices of preconditioner-solver combinations available in the PETSc library into a handful of promising candidates, selected mostly based on solver runtime. Given the preliminary nature of this study, it was performed in an in-house Linux server at the Laboratório de Simulação em Energia e Fluidos (LASEF) instead of at a proper HPC system. As such, the resulting runtimes values should not to be taken as absolute, but instead used as an order-of-magnitude estimates – which is sufficient for the desired comparison. The server – codenamed Hopper – consists of two 8-core Intel Xeon E5-2650 processors running at 3 GHz.

Table 3. List of PETSc acronyms and full names of preconditioners and solvers considered.

PETSc	Name
BJACOBI	Block-Jacobi
ASM	Additive Schwarz Method
GAMG	Geometric algebraic multigrid
CG	Conjugate gradient
BICG	Biconjugate gradient
BCGS	Biconjugate gradient stabilized
IBCGS	Improved biconjugate gradient stabilized
GMRES	Generalized minimal residual
FGMRES	Flexible generalized minimal residual
TFQMR	Transpose-free quasi-minimal residual

The results of solver runtime (including preconditioner setup time) for the FLS2 scheme are shown in Table 1 (serial) and Table 2 (parallel, $N_P = 8$), where a dash indicates that either the combination is invalid or the system did not converge. The entries for the additive Schwarz method (ASM) and geometric algebraic multigrid (GAMG) preconditioners have divisions that represent the additional parameters of overlap and threshold percentage, respectively. As shown, the choice of GAMG preconditioner considerably reduces the runtime by orders-of-magnitude in relation to Block Jacobi and ASM. However, this preconditioner resulted in a stable method only for lower threshold values (≤ 0.02). In relation to solvers, those which formed a valid combination with the GAMG preconditioner exhibit equivalent runtimes, with the exception of an outlier for CG. It is also worth noting the excellent values yielded by the combination of ASM-BiCG.

The previous results with FLS2 showed the promising combinations to be ASM with BiCG and

Table 4. Preconditioner-solver combinations for FLS8, runtime in seconds

	ASM			
	0	1	2	3
serial, BICG	10629	11867	10657	12233
parallel, BICG	2155	-	-	1816

	serial, GAMG				parallel, GAMG			
	0	2	4	6	0	2	4	6
BCGS	-	820	1337	-	-	154	213	-
GMRES	-	770	1298	-	-	165	210	-
FGMRES	-	961	1579	-	-	180	283	-
TFQMR	-	789	1290	-	-	148	206	-

GAMG with any valid solver except CG. As such, to determine if any of these good results were specific to the FLS2 case (e.g. due to the thin matrix bandwidth), additional tests using the FLS8 scheme were performed. The results are shown in Table 4, where a dash indicates that the solution did not converge. The ASM-BiCG combination performs poorly in comparison to the GAMG combinations. On the other hand, the GAMG threshold value of zero is now unstable.

The previous results indicate that, for the given problem, the best choice of preconditioner is GAMG with a threshold value of 0.02. Since the solvers analyzed in combination with GAMG yielded equivalent results, the BiCGSTAB solver was chosen due to its tradition among the CFD community.

5. Parallel Efficiency

The schemes' parallel performance were determined using results from strong- and weak-scaling tests. These two scalability measurements differ in relation to the evolution of the problem size as the number of processes increase. In strong scaling, the problem size is kept fixed throughout the test; in weak scaling, it is the workload (problem size per process) that is kept fixed. While both metrics provide useful information, the weak-scaling tests better simulate engineering situations where more resources are allocated to larger problems.

The main issue encountered in executing these tests was the fact that, for high-order methods, the memory requirement is quite restrictive. This is not a problem when running the code in parallel, since this memory cost is distributed among several machines. However, when running in a single machine, the available memory was sometimes insufficient for the complete problem. This meant that a full scaling test with high-order methods was only possible for smaller problem sizes. As such, the chosen starting mesh size for all schemes was $N_C^{\text{ref}} = 80^3$, with the mesh size evolution for weak-scaling tests listed in Table 5.

These tests were performed at the Oblivion supercomputer, which is the newest center of the Rede Nacional de Computação Avançada – the Portuguese HPC network [26]. Managed by Universidade de Évora, this supercomputer consists of 58 nodes, each with two sockets containing 18-core Intel Xeon Gold 6154 processors running at 3 GHz, connected via Infiniband EDR. This entails a peak theoretical performance of ~240 Tflop/s.

The results of selected strong and weak-scaling tests using $N_C^{\text{ref}} = 80^3$ are shown in Figure 4 as plots of solver runtime (SRT), with the corresponding efficiency listed and the dashed lines showing ideal scaling. As expected, for the same grid size higher-order methods exhibit a higher SRT due to a higher number of nonzeros in the matrix, albeit also yielding a lower numerical error. Also, the deviation from ideal scaling increases with the number of processes; this seems to stabilize for $N_p \geq 4$.

For the strong-scaling tests, a significant improvement of efficiency can be observed for 16 and 32 processes, especially for the second- and fourth-order schemes. This is due to an increase in cache performance from the additional cache memory of more systems. This advantage starts to disappear with 64 processes (spread among 32 nodes, or 64 sockets), when the socket bandwidth starts to limit the computation rate. Such decrease in efficiency is more evident with $N_p = 128$, because there is no additional L1 cache relative to $N_p = 64$ since no more sockets are included. This improved cache performance does not happen for higher-order methods (at the considered problem size) because of their higher memory requirement; if a smaller problem size were considered, the same trend would also be observed for the sixth- and eighth-order methods.

As seen by the listed values of weak scaling, the higher-order methods tend to exhibit slightly higher weak-scaling efficiencies of close to 5% higher, which shows a marginal advantage for these methods. Although this is a very promising result, one could argue that the decreased efficiency of the lower-order method stem not from an inherently worse parallel suitability of FLS2, but instead of its lower SRT. This happens because, given a lower runtime,

Table 5. Grid size N_C for a number of processors N_p, in weak-scaling tests with reference N_C^{ref}.

Grid Sizes (weak scaling, $N_C^{\text{ref}} = 80^3$)							
N_p	2	4	8	16	32	64	128
$N_C^{1/3}$	101	127	160	202	254	320	403

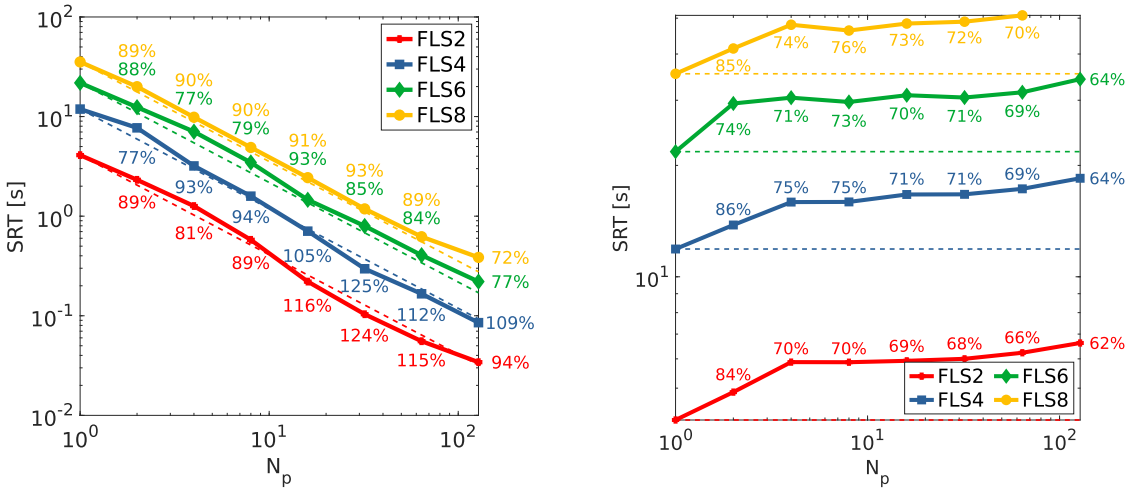

Figure 4. Solver runtime SRT for strong scaling (left) and weak scaling (right), using a reference size $N_C = 80^3$. Dashed lines indicate the ideal scaling; listed values indicate the obtained parallel efficiencies.

communication and parallel overhead costs have a higher negative effect on the overall parallel performance.

Therefore, another batch of strong- and weak-scaling tests were performed using, instead of a reference size, the same reference SRT for the initial serial execution independently of the numerical scheme. The chosen reference SRT for the strong-scaling tests was $T_s \sim 40\,\text{s}$, while that for weak-scaling tests was $T_s \sim 20\,\text{s}$. The corresponding grid sizes for each test and scheme are listed on Table 6.

The results obtained from the aforementioned tests are plotted in Figure 5. Using this reference base, the FLS2 scheme exhibits the best parallel performance, with $\eta^{st} \sim 90\%$ and $\eta^{wk} \sim 80\%$. The FLS8 also exhibits an excellent strong-scaling efficiency of $\eta^{st} \sim 90\%$, but has a lower weak-scaling efficiency that is on par with the other methods. The FLS6 exhibited the worse strong-scaling efficiency, with values falling below 70%. Even though,

Table 6. Grid sizes N_C given N_p processors, in strong- and weak-scaling tests with reference SRTs of $T_s \sim 40\,\text{s}$ and $T_s \sim 20\,\text{s}$, respectively.

Test Type	N_p	Grid Size N_C			
		FLS2	FLS4	FLS6	FLS8
Strong Scaling	-	160^3	127^3	101^3	80^3
Weak Scaling	1	127^3	101^3	80^3	63^3
	2	160^3	127^3	101^3	80^3
	4	202^3	160^3	127^3	101^3
	8	254^3	202^3	160^3	127^3
	16	320^3	254^3	202^3	160^3
	32	403^3	320^3	254^3	202^3
	64	508^3	403^3	320^3	254^3
	128	640^3	508^3	403^3	320^3

for this comparison, the FLS2 scheme performed best, one could also argue that it was because of the larger problem size used for it, which could result in a better load balancing between processes.

It is important to discuss the observed drop in parallel performance during the weak-scaling tests for values of $N_p > 32$. This happens because the Oblivion supercomputer has 58 nodes, meaning that for $N_p \geq 64$, there will be processes sharing a node and all its related infrastructure: cache, memory, bandwidth, etc. This will inevitably lead to a decrease in performance, which only worsens as the number of processes increase. Nonetheless, one may provide an estimate to the SRT value if these later runs were to have been executed with a one-to-one mapping of processes to nodes. Let $T(N_C, N_p, N_n)$ be the SRT for an N_C-sized problem executed with N_p processes spread out among N_n nodes. Then, the loss of efficiency in running the problem with a k-fold higher process-per-node concentration may be estimated by the ratio

$$\frac{T(N_C, N_p, kN_n)}{T(N_C, N_p, N_n)}\,. \tag{21}$$

Therefore, the SRT of a large problem if it were run with a one-to-one mapping of processes to nodes may be estimated using the aforementioned ratio constructed from SRTs of a smaller problem using less computation resources. For example, for the case of $N_p = 64$, the estimated SRT using this correction may be given by the following expression,

$$T(N_C, 64, 64) \approx T(N_C, 64, 32)\,\frac{T(N_C^{\text{ref}}, 2, 1)}{T(N_C^{\text{ref}}, 2, 2)}\,. \tag{22}$$

Using this method, the estimated strong- and weak-scaling efficiencies for 64 and 128 processes

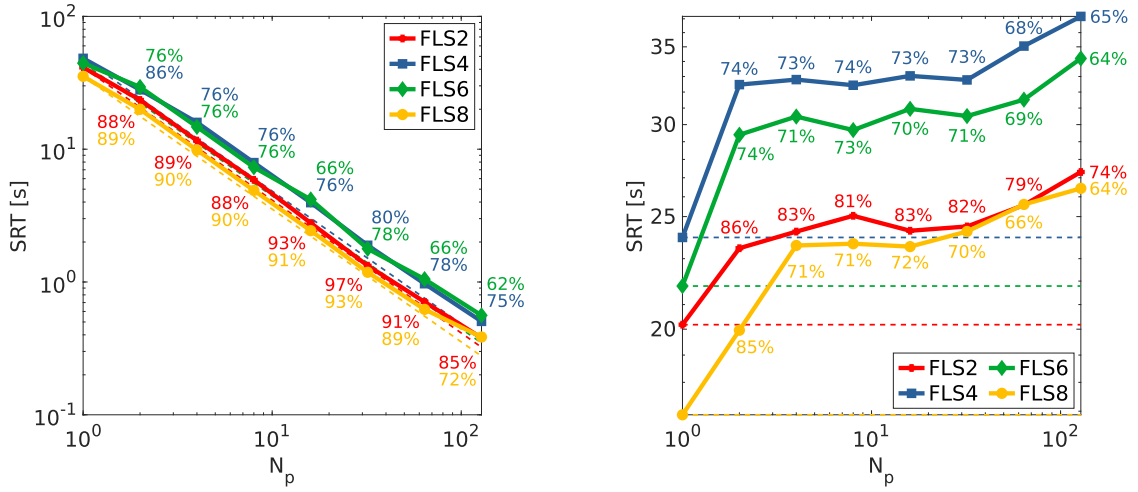

Figure 5. Solver runtime SRT for strong scaling (left) and weak scaling (right), using reference SRT of T_s~40 s and T_s~20 s, respectively. Dashed lines indicate ideal scaling; parallel efficiencies are listed.

are listed in Table 7. These estimates better continue the efficiency trend from previous N_p values, without having an acute drop in efficiency.

Table 7. Estimated weak-scaling parallel efficiency η^{wk} for high N_p values, using the correction method described by (21),(22).

Reference base	N_p	η^{wk} [%]			
		FLS2	FLS4	FLS6	FLS8
Size	64	69	74	72	73
	128	67	71	68	-
SRT	64	77	70	68	72
	128	82	72	72	70

6. Computational Cost

Lastly, it is worthwhile to compare the computational cost of each FLS method. Figure 6 plots the numerical error norm $\|\boldsymbol{\varepsilon}\|_1$ of each method as function of two different metrics for computational cost. On the left, it is plotted against the number of nonzeros N_{nz} of the corresponding coefficient matrix, which is an indirect measurement of the memory required to store the matrix. On the right, it is plotted as function of the mean SRT measured in CPU time ($N_p \times T$).

The figure above clearly shows the computational advantage of higher-order methods. For a given memory or time constraint, higher-order methods yield significantly more accurate results. This may also be seen through another manner: to obtain a given accuracy, a higher-order method requires significantly less memory and time. Both of these are important budget constraints of computational projects, so having methods that are more memory- and time-efficient is a considerable advantage.

7. Conclusion

The FLS schemes previously developed by Vasconcelos [27] and Diogo [23] were implemented in parallel for HPC and expanded to 3D problems. The resulting algorithm was verified using the previous Matlab code and also using manufactured analytical solutions based on sinusoidal functions. The influence of the floating-point format used was also analyzed, with the quadruple-precision format providing a significant increase in accuracy for finer meshes in the sixth- and eighth-order schemes. This improvement was achieved by delaying the onset of major contamination by round-off errors in the results. Therefore, an equivalent improvement can also be obtained for second- and fourth-order schemes, given sufficiently fine meshes.

In preparation for the parallel scalability tests, a comparison study of preconditioner-solver combinations was performed at an in-house machine from LASEF. The criteria for choosing the combination to be used in the scalability tests were that of fastest runtime (serial and parallel) and robustness. The choice of GAMG preconditioner was clear from the obtained results, which showed it vastly outperforming its counterparts in both serial and parallel runtime. The choice of solver was less obvious, with four solvers performing the best with not much difference between them, when paired with the GAMG preconditioner. As such, the Author chose to use the BiCGSTAB solver due to its long-standing tradition, especially within the CFD community.

The parallel performance of the developed algorithm was analyzed for three-dimensional problems at the Oblivion supercomputer managed by the HPC Center of Universidade de Évora. A series of strong- and weak-scaling tests were performed to measure scalability, with both mesh size and solver

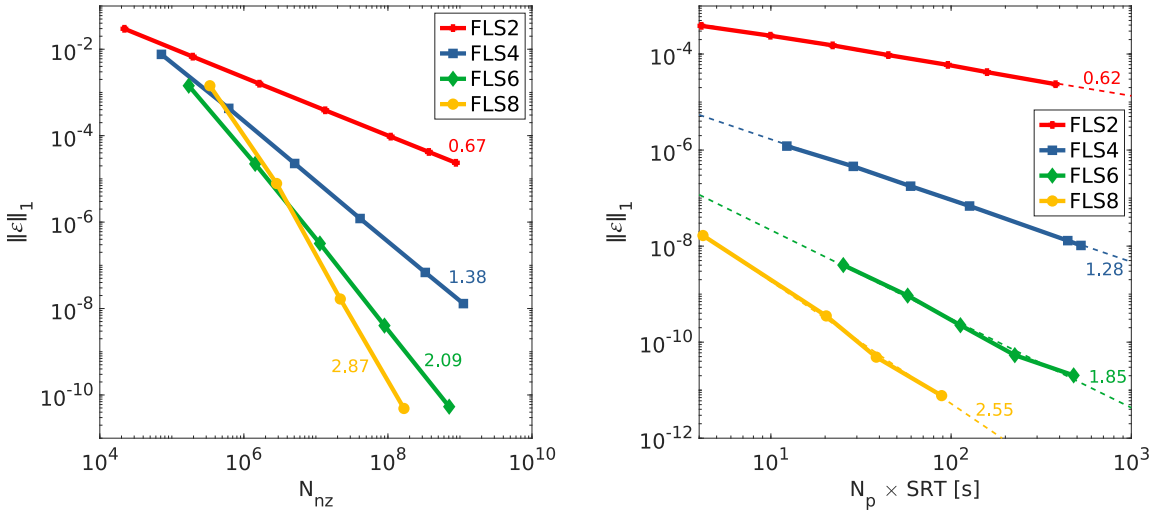

Figure 6. Numerical error norm $\|\varepsilon\|_1$ as function of number of nonzeros N_{nz} (left) and solver runtime SRT given in CPU time (right). The listed values represent the decay order of $\|\varepsilon\|_1$.

runtime fixed as reference points, thus yielding a total of four performance tests for comparison between each FLS scheme. No one scheme presented a significant advantage over others across all tests, with values for strong- and weak-scaling efficiencies of over, respectively, 85% and 70% being achieved consistently. Overall, the obtained results show that high-order methods, even with their increased number of halo cells for communication, may indeed have a parallel performance that is on par with low-order methods. Thus, this provides evidence to refute a major argument against the use of high-order methods in HPC.

Furthermore, the computational cost of low- and high-order FLS schemes were compared. This cost was measured in terms of both memory and runtime requirements for a given accuracy. Using either of these metrics, the advantage of high-order methods regarding cost efficiency was clearly demonstrated and, thus, presents a major incentive to their use over low-order methods.

References

[1] Patankar SV. *Numerical Heat Transfer and Fluid Flow*. Hemisphere, 1980. ISBN: 0-89116-522-3.

[2] Slotnick JP et al. *CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences*. Contractor Report NASA/CR-2014-218178. National Aeronautics and Space Administration, 2014. URL: https://ntrs.nasa.gov/citations/20140003093.

[3] Zingg D, De Rango S, Nemec M, Pulliam T. "Comparison of Several Spatial Discretizations for the Navier-Stokes Equations". In: *Journal of Computational Physics* 160.2 (2000), pp. 683–704. ISSN: 0021-9991. DOI: 10.1006/jcph.2000.6482.

[4] Visbal MR, Gaitonde DV. "On the Use of Higher-Order Finite-Difference Schemes on Curvilinear and Deforming Meshes". In: *Journal of Computational Physics* 181.1 (2002), pp. 155–185. ISSN: 0021-9991. DOI: 10.1006/jcph.2002.7117.

[5] Mosedale A, Drikakis D. "Assessment of Very High Order of Accuracy in Implicit LES models". In: *Journal of Fluids Engineering* 129.12 (2007), pp. 1497–1503. ISSN: 0098-2202. DOI: 10.1115/1.2801374.

[6] Ekaterinaris JA. "High-order accurate, low numerical diffusion methods for aerodynamics". In: *Progress in Aerospace Sciences* 41.3 (2005), pp. 192–300. ISSN: 0376-0421. DOI: 10.1016/j.paerosci.2005.03.003.

[7] Barth T, Frederickson P. "Higher order solution of the Euler equations on unstructured grids using quadratic reconstruction". In: *28th Aerospace Sciences Meeting*. DOI: 10.2514/6.1990-13.

[8] Ollivier-Gooch CF, Van Altena M. "A High-Order-Accurate Unstructured Mesh Finite-Volume Scheme for the Advection-Diffusion Equation". In: *Journal of Computational Physics* 181.2 (2002), pp. 729–752. ISSN: 0021-9991. DOI: 10.1006/jcph.2002.7159.

[9] Nejat A, Ollivier-Gooch CF. "A high-order accurate unstructured finite volume Newton-Krylov algorithm for inviscid compressible flows". In: *Journal of Computational Physics* 227.4 (2008), pp. 2582–2609. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2007.11.011.

[10] Michalak K, Ollivier-Gooch CF. "Limiters for Unstructured Higher-Order Accurate Solutions of the Euler Equations". In: *46th AIAA Aerospace Sciences Meeting and Exhibit*. DOI: 10.2514/6.2008-776.

[11] Clain SL, Machado GJ, Pereira RMS. "A new very high-order finite volume method for the 2D convection-diffusion problem on unstructured meshes". In: *IV Confererência Nacional em Mecânica dos Fluidos, Termodinâmica e Energia*. 2012. HAL: hal-00675743.

[12] Costa R, Clain S, Machado GJ. "New cell-vertex reconstruction for finite volume scheme: Application to the convection-diffusion-reaction equation". In: *Computers & Mathematics with Applications* 68.10 (2014), pp. 1229–1249. ISSN: 0898-1221. DOI: 10.1016/j.camwa.2014.08.015.

[13] Costa R, Clain S, Machado GJ. "A sixth-order finite volume scheme for the steady-state incompressible Stokes equations on staggered unstructured meshes". In: *Journal of Computational Physics* 349 (2017), pp. 501–527. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2017.07.047.

[14] Wang Z. "High-order methods for the Euler and Navier-Stokes equations on unstructured grids". In: *Progress in Aerospace Sciences* 43.1 (2007), pp. 1–41. ISSN: 0376-0421. DOI: 10.1016/j.paerosci.2007.05.001.

[15] Ollivier-Gooch CF. "High-order ENO schemes for unstructured meshes based on least-squares reconstruction". In: *35th AIAA Aerospace Sciences Meeting and Exhibit*. 1997. DOI: 10.2514/6.1997-540.

[16] Friedrich O. "Weighted Essentially Non-Oscillatory Schemes for the Interpolation of Mean Values on Unstructured Grids". In: *Journal of Computational Physics* 144.1 (1998), pp. 194–212. ISSN: 0021-9991. DOI: 10.1006/jcph.1998.5988.

[17] Hu C, Shu C-W. "Weighted Essentially Non-oscillatory Schemes on Triangular Meshes". In: *Journal of Computational Physics* 150.1 (1999), pp. 97–127. ISSN: 0021-9991. DOI: 10.1006/jcph.1998.6165.

[18] Ivan L, Groth CP. "High-order solution-adaptive central essentially non-oscillatory (CENO) method for viscous flows". In: *Journal of Computational Physics* 257 (2014), pp. 830–862. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2013.09.045.

[19] Fu L, Hu XY, Adams NA. "A new class of adaptive high-order targeted ENO schemes for hyperbolic conservation laws". In: *Journal of Computational Physics* 374 (2018), pp. 724–751. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2018.07.043.

[20] Cueto-Felgueroso L, Colominas I, Fe J, Navarrina F, Casteleiro M. "High-order finite volume schemes on unstructured grids using moving least-squares reconstruction. Application to shallow water dynamics". In: *International journal for numerical methods in engineering* 65.3 (2006), pp. 295–331.

[21] Cueto-Felgueroso L, Colominas I, Nogueira X, Navarrina F, Casteleiro M. "Finite volume solvers and Moving Least-Squares approximations for the compressible Navier-Stokes equations on unstructured grids". In: *Computer Methods in Applied Mechanics and Engineering* 196.45 (2007), pp. 4712–4736. ISSN: 0045-7825. DOI: 10.1016/j.cma.2007.06.003.

[22] Moukalled F, Mangani L, Darwish M. *The Finite Volume Method in Computational Fluid Dynamics. An Advanced Introduction with OpenFOAM® and Matlab®*. Springer, 2016. ISBN: 978-3-319-16874-6. DOI: 10.1007/978-3-319-16874-6.

[23] Diogo FJM. "A Very High-Order Finite Volume Technique for Convection-Diffusion Problems on Unstructured Grids". MSc thesis. Instituto Superior Técnico, Universidade de Lisboa, 2019. FENIX: 283828618790445 [MEAER].

[24] Goldberg D. "What Every Computer Scientist Should Know about Floating-Point Arithmetic". In: *ACM Comput. Surv.* 23.1 (1991), pp. 5–48. ISSN: 0360-0300. DOI: 10.1145/103162.103163.

[25] Balay S et al. *PETSc/TAO Users Manual*. Tech. rep. ANL-21/39 - Revision 3.17. Argonne National Laboratory, 2022.

[26] *High Performance Computing da Universidade de Évora*. Rede National de Computação Avançada. URL: https://rnca.fccn.pt/hpc-ue/ (visited on 07/2022).

[27] Vasconcelos AGR. "A Very High-Order Finite Volume Method Based on Weighted Least Squares for the Solution of Poisson Equation on Unstructured Grids". MSc thesis. Instituto Superior Técnico, Universidade de Lisboa, 2017. FENIX: 1972678479053984 [MEAER].