Advantages of static condensation in implicit compressible Navier-Stokes DGSEM solvers

Wojciech Laskowski[a], Andrés M. Rueda-Ramírez[ac], Gonzalo Rubio[ab], Eusebio Valero[ab], Esteban Ferrer[ab]

[a] *ETSIAE-UPM (School of Aeronautics - Universidad Politécnica de Madrid) - Plaza de Cardenal Cisneros 3, 28040 Madrid, Spain*

[b] *Center for Computational Simulation - Universidad Politécnica de Madrid, Campus de Montegancedo, Boadilla del Monte, 28660 Madrid, Spain*

[c] *Department for Mathematics and Computer Science, University of Cologne, Weyertal 86-90, 50931, Cologne, Germany*

Abstract

We consider implicit time-marching schemes for the compressible Navier-Stokes equations, discretised using the Discontinuous Galerkin Spectral Element Method with Gauss-Lobatto nodal points (GL-DGSEM). We compare classic implicit strategies for the full Jacobian system to our recently developed static condensation technique for GL-DGSEM *Rueda-Ramírez et al. (2019), A Statically Condensed Discontinuous Galerkin Spectral Element Method on Gauss-Lobatto Nodes for the Compressible Navier-Stokes Equations* [1]. The Navier-Stokes system is linearised using a Newton-Raphson method and solved using an iterative preconditioned-GMRES solver. Both the full and statically condensed systems benefit from a Block-Jacobi preconditioner.

We include theoretical estimates for the various costs involved (i.e. calculation of full and condensed Jacobians, factorising and inverting the preconditioners, GMRES steps and overall costs) to clarify the advantages of using

static condensation in GL-DGSEM, for varying polynomial orders. These estimates are then examined for a steady three-dimensional manufactured solution problem and for an two-dimensional unsteady laminar flow over a NACA0012 airfoil. In all cases, we test the schemes for high polynomial orders, which range from 2 to 8 for a manufactured solution case and from 2 to 5 for the NACA0012 airfoil. The statically condensed system shows computational savings, which relate to the smaller system size and cheaper Block-Jacobi preconditioner with smaller blocks and better polynomial scaling, when compared to the preconditioned full Jacobian system (not condensed). The advantage of using static condensation is more noticeable for higher polynomial orders.

Keywords:

High-order discontinuous Galerkin, DGSEM, Gauss-Lobatto, Implicit time-marching, preconditioned-GMRES, Compressible Navier-Stokes, Static condensation, NACA0012 airfoil

---

Contents

1. Introduction

The accurate simulation of aerodynamic characteristics over lifting surfaces (airfoils and wings) is of major importance to the aeronautical industry and can potentially reduce fuel consumption by allowing lighter aircraft designs. High order methods, and particularly discontinuous Galerkin (DG)

schemes, are well equipped to provide high accuracy on coarse meshes due to their spectral convergence property (i.e. exponential decay of the error). In the last decade, these methods have gained popularity for solving fluid flows governed by the incompressible, e.g. [2, 3, 4, 5, 6] and compressible Navier-Stokes equations, e.g. [7, 8, 9, 10]. DG solutions show improved accuracy over low order methods, but are often expensive to compute [11]. In recent years, acceleration techniques for DG schemes have focused on local p-adaption, see e.g. [7, 10] and on improved time-marching techniques, e.g. FAS p-multigrid [10], that allow for faster convergence and large time-steps, with important savings in computational cost.

The Discontinuous Galerkin Spectral Element Method (DGSEM) [12], is a particular nodal version of DG, which has proved to be very efficient on hexahedral elements (e.g. diagonal mass matrices). Additionally, the variant of the DGSEM where Gauss-Lobatto nodal points are selected, i.e. GL-DGSEM, is well suited for the development of provably stable schemes [13], fulfilling the summation-by-parts property [14]. These schemes have enhanced stability and are convenient for under-resolved simulations, if split-forms of the governing equations are discretised. Examples of provably stable formulations can be found for the Euler [14], the Magneto-Hydrodynamics [15], multiphase flows [16, 17] and the Navier-Stokes equations [18, 9, 19].

We have recently shown an additional advantage of GL-DGSEM [1]: it is well suited for the static condensation approach, whilst the classic Gauss point version is not. In this work we exploit the statically condensed system,

to accelerate implicit time advancement with and iterative GMRES solver, and compare the accelerations to the traditional full Jacobian system. Note that both approaches rely on Newton-Raphson linearisation to obtain the full and condensed systems. In this work, we do not include split-forms but propose a static condensation technique, which is perfectly applicable to formulations including stabilising split-forms (e.g. two point fluxes), and may be combined with the static condensation, in future work.

Static condensation has been widely applied in the context of high order methods, and is a popular strategy in the continuous Galerkin community, e.g. [20, 21], where it has proved to be an efficient strategy to solve large systems in both structural and fluid mechanics, e.g. [20, 22]. Static condensation can be combined with modern iterative techniques such as p-multigrid with domain decomposition smothers tailored for condensed systems [23]. Recently, Pardo et al. [24] showed that static condensation proves beneficial when combined with iterative solvers, if the number of iterations is sufficiently large, to compensate for the additional cost associated of computing the system's Schur complement. Similar findings are included in this work for DGSEM.

Static condensation has been applied to discontinuous Galerkin discretisations by Sherwin et al. [25] and Hybridized Discontinuous Galerkin (HDG), e.g. [26, 27, 28]. In the first work, Sherwin et al. reported advantages of statically condensed systems when using tailored non-orthogonal basis functions (i.e. non-diagonal mass matrices). The remaining references were developed

for HDG formulations, where the method decouples the degrees of freedom belonging to the mesh elements from the mesh skeleton, enabling static condensation. However, HDG requires specific numerical fluxes [1, 26, 29], restricting the use of well known Riemann approximations, such as Roe's. Our static condensation for GL-DGSEM allows any flux.

In our previous work [1], we showed the detailed implementation of the static condensation approach in GL-DGSEM, and applied the method to solve steady cases using direct solvers and an implicit GMRES with a point-Jacobi preconditioner. In this work, we extend that analysis further by comparing the performance of statically condensed and full Jacobian (non-condensed) systems for Block-Jacobi preconditioner in steady and unsteady problems, and show that the statically condensed system can lead to faster iterative GMRES solves. We include theoretical estimates to analyse and extrapolate the costs involved with respect to the polynomial order. These include the calculation of full and condensed Jacobians, the factorisation and invertion of the preconditioner and the preconditioned-GMRES steps. Additionally, we briefly asses the use of ILU(k) preconditioners and include a section to verify that the advantages of the statically condensed GL-DGSEM are essentially independent of the Mach and Reynolds numbers.

Both full and condensed systems can benefit from preconditioners to accelerate convergence. Efficient preconditioners should be cheap to construct and to parallelise, whilst enhancing the convergence of the system, e.g. reducing the number of iterations to reach convergence. Iterative strate-

gies (including preconditioners) for DG discretisations of both compressible and incompressible flows have been widely explored in recent years [30, 31, 32, 33, 34, 35, 36, 37, 28, 38, 39]. Most authors employ block structured preconditioners/p-multigrid smothers, such as Block-Jacobi, Line-Jacobi, additive-Schwarz or Block-ILU. Among these, [37, 28, 39] focused on coarse grid accelerations and efficient implementation of the *state-of-the-art* solvers for turbulent problems, which is out of the scope of this work. Point ILU has also been successfully used for aerodynamic applications in [40, 41]. Persson and Peraire [32] or Gopalakrishnan and Kanschat [42] showed that element-block based preconditioners are essential to eliminate high p dependent errors. It is also very natural to exploit the element-block structure of the Jacobian (specially in the parallel computations due to the block locality that enables to perform block inversions locally), as most of these methods require the direct factorisation of block matrices. Note that this can become troublesome for high polynomial orders, especially in three-dimensional flows. In this work, we select Block-Jacobi preconditioner and show that when condensing the system, the preconditioner scales more gently for high polynomials, than the preconditioner for the full system. This translates into lower costs for all the steps where the preconditioner is required (i.e. factorisation of the blocks and GMRES step involving the preconditioner), and paves the way to using high polynomial orders efficiently.

Our comparisons are novel in that the static condensation technique, recently developed for GL-DGSEM by the authors, is directly challenged to

the state of the art implicit preconditioned-GMRES solvers to show computational savings for steady and unsteady flows and a range of polynomial order ranging from 2 to 8. The results are backed-up by the theoretical estimates for the various costs. The beneficial effect of statically condense the system is observed for various Mach and Reynolds numbers, suggesting that this technique can be exploited for a wide range of flow regimes in steady and unsteady flows.

In what follows, we describe the methodology with emphasis on the time marching scheme and implementation details. We continue with the theoretical estimates and the simulations, where we compare the full Jacobian and the static condensation for a 3D Manufactured Solution problem and the unsteady flow over a 2D NACA0012 airfoil. We finalise with conclusions and outlooks.

2. Methodology

We use the nodal Discontinuous Galerkin Spectral Element Method (DGSEM) introduced by Black [43], where the computational domain is tessellated into non-overlapping hexahedral elements. In the DGSEM, numerical fluxes are necessary to transfer information between discontinuous element solutions. Here, we retain Lax-Friedrichs fluxes for the convective fluxes and the Interior Penalty method for viscous fluxes, but other fluxes with compact support could also be used (e.g. Roe for convection or BR2 for diffusion). The selected fluxes yield a compact mesh stencil and are differentiated to obtain

an analytical Jacobian. Further details on how the Jacobian can be obtained along with the peculiarities and sparsity patterns resulting from using Gauss-Lobatto nodal points, can be found in our previous works [1, 44].

2.1. Time-implicit discretisation and Jacobian computation

Let us briefly describe the implicit methods retained in this work. After discretising the compressible Navier-Stokes equations, we obtain the following system of equations

$$\underline{\boldsymbol{M}}\frac{\partial \boldsymbol{Q}}{\partial t} + \boldsymbol{F}(\boldsymbol{Q}) = \underline{\boldsymbol{M}}\boldsymbol{S}, \tag{1}$$

where $\boldsymbol{Q}$ is a vector that stores the conservative variables in all degrees of freedom of the domain, $\boldsymbol{F}(\boldsymbol{Q})$ encompasses both discrete convective and diffusive fluxes, $\underline{\boldsymbol{M}}$ is the mass matrix, which is diagonal in the nodal DGSEM approach, and $\boldsymbol{S}$ is a source term.

We replace the continuous in time derivative in (1) by a discrete implicit time integration scheme using Backward Differentiation Formulas (BDF) of order 1 and 2 (BDF1 or BDF2),

$$\frac{\partial \boldsymbol{Q}}{\partial t} \leftarrow \frac{\delta \boldsymbol{Q}}{\delta t}(\boldsymbol{Q}_{s+1}, \boldsymbol{Q}_s, \cdots), \tag{2}$$

where the operator $\delta\boldsymbol{Q}/\delta t$ is a function of the solution on the next time step, $\boldsymbol{Q}_{s+1}$ (the unknown), the current time step, $\boldsymbol{Q}_s$, and possibly previous time steps. When treated implicitly, the nonlinear operator $\boldsymbol{F}$, in equation (1) is

evaluated for the unknown solutions, $\boldsymbol{Q}_{s+1}$. Considering this, equation (1) can then be rewritten as

$$\boldsymbol{R}(\boldsymbol{Q}_{s+1}) = \frac{\delta \boldsymbol{Q}}{\delta t}(\boldsymbol{Q}_{s+1}, \boldsymbol{Q}_s, \cdots) + \underline{\boldsymbol{M}}^{-1}\boldsymbol{F}(\boldsymbol{Q}_{s+1}) - \boldsymbol{S} = \boldsymbol{0}. \tag{3}$$

Note that in the DGSEM approach the mass matrix $\underline{\boldsymbol{M}}$ is diagonal and can be trivially inverted, leading to an efficient discontinuous Galerkin method. When computing steady flows, we are not interested in producing an accurate solution in time, and therefore we use an implicit BDF of order 1 to advance until steady state. However, for unsteady cases we will use an implicit BDF of order 2 and shorter time steps to obtain accurate solutions in time.

The nonlinear system of equations, (3), can be solved using Newton-Raphson iterations to obtain the linear system:

$$\underline{\boldsymbol{A}}\Delta\boldsymbol{Q} = \boldsymbol{B}, \tag{4}$$

where $\underline{\boldsymbol{A}} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{Q}}(\tilde{\boldsymbol{Q}}_{s+1})$ is the Jacobian matrix evaluated at $\tilde{\boldsymbol{Q}}_{s+1}$, which is an approximation to the unknown solution $\boldsymbol{Q}_{s+1}$. The right-hand-side is $\boldsymbol{B} = -\boldsymbol{R}(\tilde{\boldsymbol{Q}}_{s+1})$. Equation (4) is a linear system that must be solved iteratively to approach $\boldsymbol{Q}_{s+1} \leftarrow \tilde{\boldsymbol{Q}}_{s+1} + \Delta\boldsymbol{Q}$. The Jacobian matrix $\underline{\boldsymbol{A}}$ may be computed analytically or numerically, and here we retain the analytical approach, for its efficiency. Equation (4) is what we refer as *full system* with $\underline{\boldsymbol{A}}$ the *full Jacobian*.

2.2. Static condensation

In the GL-DGSEM framework, we can statically condense system (4) to obtain the following form

$$\begin{bmatrix} \underline{\boldsymbol{A}}_{bb} - \underline{\boldsymbol{A}}_{ib}\underline{\boldsymbol{A}}_{ii}^{-1}\underline{\boldsymbol{A}}_{bi} & \boldsymbol{0} \\ \underline{\boldsymbol{A}}_{bi} & \underline{\boldsymbol{A}}_{ii} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{Q}_b \\ \Delta \boldsymbol{Q}_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{B}_b - \underline{\boldsymbol{A}}_{ib}\underline{\boldsymbol{A}}_{ii}^{-1}\boldsymbol{B}_i \\ \boldsymbol{B}_i \end{bmatrix}, \tag{5}$$

where subindex b and i denote boundary and interior nodes, respectively. The main interest of the method is to obtain a block diagonal matrix $\underline{\boldsymbol{A}}_{ii}$, that can be inverted cheaply and locally (element by element). Additionally, the boundary matrix including the degrees of freedom linking boundaries between elements, is greatly reduced by the use of Gauss-Lobatto points in DGSEM [1]. The resulting system is equivalent to the full system, but can be decoupled in two subsystems. The first one for the skeleton of the mesh, our *condensed system of equations* is

$$\underline{\boldsymbol{A}}_{cond}\Delta \boldsymbol{Q}_b = \boldsymbol{B}_{cond}, \tag{6}$$

where $\underline{\boldsymbol{A}}_{cond} = \underline{\boldsymbol{A}}_{bb} - \underline{\boldsymbol{A}}_{ib}\underline{\boldsymbol{A}}_{ii}^{-1}\underline{\boldsymbol{A}}_{bi}$ and $\boldsymbol{B}_{cond} = \boldsymbol{B}_b - \underline{\boldsymbol{A}}_{ib}\underline{\boldsymbol{A}}_{ii}^{-1}\boldsymbol{B}_i$. Once the condensed system (6), based on the Schur complement $\underline{\boldsymbol{A}}_{cond}$, is solved, then it is trivial to substitute and solve for the second system $\Delta \boldsymbol{Q}_i = \underline{\boldsymbol{A}}_{ii}^{-1}(\boldsymbol{B}_i - \underline{\boldsymbol{A}}_{bi}\Delta \boldsymbol{Q}_b)$, since $\underline{\boldsymbol{A}}_{ii}$ is block diagonal and has already being factorised to compute $\underline{\boldsymbol{A}}_{cond}$.

2.3. Size of the full and the condensed Jacobians

One of the main advantages of the static condensation is the reduced size of the matrix $\underline{\boldsymbol{A}}_{cond}$ (with only the mesh skeleton degrees of freedom) in comparison with the original Jacobian matrix $\underline{\boldsymbol{A}}$ (with all the degrees of freedom in the mesh). We can quantify the number of degrees of freedom for our GL-DGSEM discretisation. The Jacobian matrix $\underline{\boldsymbol{A}}$ has size

$$n = N_{el} \cdot nb. \tag{7}$$

where N_{el} is number of elements and nb is the size of each element-block. Then, assuming mesh elements with isotropic polynomial order P, we can describe the size of each block nb as a function of P, the dimension d (e.g. $d = 3$ for 3D meshes) and the number of conservative variables (or equations) in the computational domain for the Navier-Stokes equations N_{eq} (e.g. $N_{eq} = 5$ in 3D):

$$nb = N_{eq}(P+1)^d. \tag{8}$$

Equation (7) can also be used to describe the size of the matrices, $\underline{\boldsymbol{A}}_{ii}$ and $\underline{\boldsymbol{A}}_{bb}$, involved in the Schur complement computation and included in the statically condensed system (5) with $n_{ii} = N_{el} \cdot nb_{ii}$ and $n_{bb} = N_{el} \cdot nb_{bb}$, with the only difference being the block sizes. Here, the block size of the element-skeleton matrix nb_{bb} directly corresponds to the size of the block of the final Schur complement $\underline{\boldsymbol{A}}_{cond}$. The blocks for the condensed matrix arise from having decoupled element interior i from the element boundary nodes

b, leaving fewer degrees of freedom per block. Thus, the size of the block of matrix $\underline{\boldsymbol{A}}_{ii}$, that corresponds to the interior of the elements is

$$nb_{ii} = N_{eq}(P-1)^d. \tag{9}$$

Consequently, the size of the block of $\underline{\boldsymbol{A}}_{bb}$ and $\underline{\boldsymbol{A}}_{cond}$ can be defined as the difference between the size of the element-block and the interior element part

$$nb_{bb} = N_{eq}\left[(P+1)^d - (P-1)^d\right], \tag{10}$$

and with these blocks, the final size of the matrices could be easily computed from equation (7).

Additionally, it is possible to obtain estimates for the number of non-zero entries nnz in the full and condensed Jacobian. This is not a trivial task, and details are included in Appendix C. The final expressions are summarised in Table 1, for 3D and 2D.

Table 1: Explicit formulas for the leading terms of block sizes, estimation of number of non-zeros nnz per block, and matrix non-zero entries, for the full and condensed systems in 2D and 3D. All provided as functions of the number of elements N_{el}, polynomial order P and number of conservative variables in the 3D domain, i.e. $N_{eq} = 5$ for the compressible Navier-Stokes equations.

3D		
	Full system	Condensed system
Block size	$N_{eq}(P+1)^3$	$N_{eq}(6P^2+2)$
nnz per block	$3N_{eq}^2P(P+1)^4$	$N_{eq}^2(6P^2+2)^2$
nnz in matrix	$3N_{el}N_{eq}^2P(P+1)^4$	$25N_{el}N_{eq}^2(6P^2+2)^2$
2D		
	Full system	Condensed system
Block size	$N_{eq}(P+1)^2$	$N_{eq}4P$
nnz per block	$N_{eq}^2(P+1)^4$	$N_{eq}^2 16P^2$
nnz in matrix	$N_{el}N_{eq}^2(P+1)^4$	$13N_{el}N_{eq}^2 16P^2$

Let us remark that the expressions for the block sizes are exact. However, the expressions for the nnz per block are upper bounds derived in the appendix. The entry corresponding to the nnz for the full system, only includes the diagonal blocks corresponding to the viscous terms, since these are asymptotically dominant, as they scale $\mathcal{O}(P^5)$ (all other blocks have weaker scaling, see appendix for details). The total number of non-zeros might be obtained multiplying by the number of elements. Regarding the condensed system, here the block stencil of this matrix is estimated to be 25 in 3D and 13 in 2D (neighbor to neighbor coupling), and therefore to obtain the total number of nnz in the matrix, the nnz per block need to be multiplied by the number of elements and by the constant (25 or 13) accounting for the neighbour coupling.

Finally, the condensed system presents smaller and denser blocks and

the block stencil of the condensed system is wider than the one of the full system. As a result, the nnz of the condensed system is larger than the one of the full system. Regarding the total number of non-zero entries in the matrix, the scalings show that the full system will asymptotically contain more non-zero entries for large polynomial orders. However, due to the denser connectivity in the condensed system, the non-zero entries can be higher for low polynomial orders.

In the Continuous Galerkin formulation for simple diffusion or advection-diffusion problems [24, 45], the number of non-zero entries in the condensed matrix decreases with respect to nnz in the full system. However, in our case for the GL-DGSEM of the compressible Navier-Stokes equations, the number of non-zeros increases. Increased number of non-zeros for the condensed system have been reported by Habchi [46], for an elastohydrodynamic lubrication problem. There, the authors considered several meshes for the same contact problem, from *extra coarse* to *extra fine*. The results show that nnz in the condensed systems is reduced for coarse meshes, whereas for the others $nnz_{cond} > nnz_{full}$.

Complementary illustrations of the static condensation sparsity patterns for the GL-DGSEM approach may be found in our previous work [1]. In this work, we concentrate on comparing the efficiency of solving the linear system of equations, i.e. solving full system (4) to solving the two subsystems for the condensed system (6) using iterative methods. To account for the iterative costs, we will use the matrix sizes and number of non-zeros, included in Table

1.

2.4. Preconditioned-GMRES solver

We use preconditioned-GMRES to sove both the full system (4), and the statically condensed system (6). Previous works [33, 31, 32, 35, 36] have shown that combining GMRES and block preconditioners is effective in solving Eq. (4) for DG discretisations of Euler, Navier-Stokes or RANS equations. Here, we have considered several preconditioning strategies, namely element Block-Jacobi and incomplete LU factorisation with different factorisation levels, ILU(k). We conduct a preliminary evaluation of these preconditioners for the full and condensed systems in Appendix A. For the manufactured solution case (to be described later in detail), ILU(k) preconditioners perform better in terms of iteration count and overall cost, but show high cost when computing the preconditioner. Block-Jacobi does not perform as well as ILU(k) in terms of overall solver cost, but provides a lower factorisation cost (specially for the statically condensed system) and provides very competitive average iteration count and average solver cost. Additionally, a Block-Jacobi preconditioner is more suitable for parallel [28] and matrix-free [47, 48] computations, since the blocks can be inverted locally whilst exploiting the block-structure of the high order DGSEM discretisation, as well as requiring less memory [33, 47]. For this reason, in the following sections, we present all results with Block-Jacobi preconditioners for both the full (4) and the statically condensed (6) systems.

The Block-Jacobi preconditioner ignores all the Jacobian off-diagonal blocks and performs a local LU decomposition (factorisation step) in each diagonal block. For the full system, these diagonal blocks include all the element degrees of freedom for each element, whilst the size for the blocks is reduced in the condensed system (only skeleton degrees of freedom): matrix $\underline{\boldsymbol{A}}_{cond}$ in (6). These blocks are smaller as shown in Table 1 and therefore constructing the Block-Jacobi preconditioner for the condensed system is much cheaper, than for the full system, and especially for high polynomial orders (for a more detailed comparison of the factorisation costs, see next Section 3).

Finally, all the operations related to the preconditioned-GMRES solver (computing preconditioner and performing GMRES iterations) are performed using the well known open-source library PETSc [49, 50, 51]. The computation of the condensed system (6), however, is done with our in-house code. Note that PETSc has been widely used in aeronautical publications, including DGSEM flow simulations [33, 52, 53]. By selecting this well validated implementation, we avoid in-house inefficiencies that could mask the outcomes of our comparisons.

2.5. Further implementation details

In the result section, we also include explicit time-marching (ESRK3) [54] simulations for reference, but comparisons of overall computing time are not of interest in this work. Indeed, it is well known, that the explicit

time integrator is easy to parallelise with appropriate domain partitioning [55, 56] and could produce very efficient solutions when using large number of processors, whilst implicit schemes require a greater effort and increased memory requirements for matrix-based solvers [31, 11]. Alternative matrix-free approaches have been proposed, e.g. Pazner and Persson [48], but are not explored in this text. For the above mentioned reasons, all cases are run in serial such that all approaches are fairly compared without taking into account parallelisation strategies or communication efficiency.

3. Theoretical costs of full and statically condensed systems

In this section, a theoretical analysis of the main computational costs of the implicit time marching scheme are included. Algorithm 1 presents the essential steps of the time marching scheme to conduct the simulation until the finalisation criteria is met. We focus only in three main steps that constitute the majority of the computational costs, i.e.:

- Step 8: cost for obtaining the statically condensed system $\underline{\boldsymbol{A}}_{cond}$,
- Step 9: cost for factorising (constructing and inverting) the preconditioning matrix $\underline{\boldsymbol{P}}^{-1}$. In the context of this work, it is the cost of factorising the element-diagonal blocks of the Jacobian system $\underline{\boldsymbol{A}}$ or the condensed system $\underline{\boldsymbol{A}}_{cond}$, which are then inverted and stored in the preconditioning matrix $\underline{\boldsymbol{P}}^{-1}$.
- Step 13: cost for solving the linear system (4) for the full system or (6)

for the condensed system, using the preconditioned-GMRES solver at each time step and as long as $||\Delta\mathbf{Q}||_\infty < \text{TOL}_{Newton}$.

Algorithm 1 Time-marching scheme including Newton-Raphson linearisation

1: $\mathbf{Q} \leftarrow$ INITIALISE()
2: **while** Steady: $||\underline{\boldsymbol{M}}^{-1}\boldsymbol{F}(\boldsymbol{Q}) - \boldsymbol{S}||_\infty < 10^{-8}$ or Unsteady: $t < T_{end}$ **do**
3: $\quad t \leftarrow t + \Delta t$
4: $\quad$ **while** $||\Delta\mathbf{Q}||_\infty < \text{TOL}_{Newton}$ **do**
5: $\quad\quad$ **if** InaccurateJacobian **then**
6: $\quad\quad\quad \underline{\boldsymbol{A}} \leftarrow$ COMPUTEFULLSYSTJACOBIAN$(\mathbf{Q}, \Delta t)$
7: $\quad\quad\quad$ **if** CondensedSystem **then**
8: $\quad\quad\quad\quad \underline{\boldsymbol{A}} \leftarrow$ COMPUTECONDENSEDJACOBIAN$(\underline{\boldsymbol{A}})$
9: $\quad\quad\quad \underline{\boldsymbol{P}}^{-1} \leftarrow$ FACTORISEPRECONDITIONER$(\underline{\boldsymbol{A}})$
10: $\quad\quad \mathbf{B} \leftarrow$ COMPUTEFULLSYSTRHS$(-\mathbf{R}(\mathbf{Q}))$
11: $\quad\quad$ **if** CondensedSystem **then**
12: $\quad\quad\quad \mathbf{B} \leftarrow$ COMPUTECONDENSEDRHS$(\underline{\boldsymbol{A}}, \mathbf{B})$
13: $\quad\quad \Delta\mathbf{Q} \leftarrow$ GMRES-SOLVE$(\underline{\boldsymbol{A}}, \underline{\boldsymbol{P}}^{-1}, \mathbf{B})$
14: $\quad\quad$ **if** CondensedSystem **then**
15: $\quad\quad\quad \Delta\mathbf{Q} \leftarrow$ COMPUTEINTERIORSOLUTION$(\underline{\boldsymbol{A}}, \Delta\mathbf{Q}, \mathbf{B})$
16: $\quad\quad \mathbf{Q} = \mathbf{Q} + \Delta\mathbf{Q}$

Step 13 solves the linear system using preconditioned-GMRES (further discussed below) and one must account for its cost in every Newton iteration and for every time step. Steps 5 to 9 need to be computed when the Jacobian matrix $\underline{\boldsymbol{A}}(\mathbf{Q}, \Delta t)$ (or the condensed version), has significantly changed, which leads to a quasi-Newton method. Naturally, re-using the Jacobian matrix from the previous time steps may inhibit quadratic convergence of the Newton-Raphson method [57]. To ensure a sufficiently high convergence rate, we follow ideas from Zahr and Persson [58] and define a condition that

secures at least 1/4 of an order of magnitude decay per Newton iteration (see step 5 of Algorithm 1). Therefore, if the aforementioned condition is met, the Jacobian $\underline{\boldsymbol{A}}$ and preconditioner $\underline{\boldsymbol{P}}^{-1}$ are still useful and are not recomputed. In all the simulations, the Newton tolerance is set to $\mathrm{TOL}_{Newton} = 10^{-5}$, which yields accurate results. Furthermore, as in Nastase and Mavriplis [59], the preconditioned-GMRES solver tolerance is set according to the maximum norm of the residual, e, such that $\mathrm{TOL}_{GMRES} = e \cdot 0.7^i$, where i is the current Newton iteration.

Sections 3.1, 3.2 and 3.3 present the estimation of the computational costs related to the static condensation (Step 8), the preconditioner factorisation (Step 9) and the GMRES solver (Step 13). Subsequently, comparisons with the simulated costs are included in Section 4, and summarised in table 2.

3.1. Cost of static condensation

The necessary operations to obtain the condensed system (6) are detailed here:

- Factorisation and inverting the block diagonal matrix representing inner-element $\underline{\boldsymbol{A}}_{ii}^{-1}$,
- Computing $\underline{\boldsymbol{A}}_{ii}^{-1}\underline{\boldsymbol{A}}_{ib}$ and assembling the RHS of the equation (6),
- Computing the $\underline{\boldsymbol{A}}_{cond} = \underline{\boldsymbol{A}}_{bb} - \underline{\boldsymbol{A}}_{bi}\underline{\boldsymbol{A}}_{ii}^{-1}\underline{\boldsymbol{A}}_{ib}$, equation (6),
- Obtaining the solution for the interior nodes: $\Delta\boldsymbol{Q}_i = \underline{\boldsymbol{A}}_{ii}^{-1}(\boldsymbol{B}_i - \underline{\boldsymbol{A}}_{ib}\Delta\boldsymbol{Q}_b)$.

All of these operations are included in one unique cost, referred to as *condensation cost*, in the following sections. These operations are performed in Step 8 in Algorithm 1. The only exceptions are obtaining the solution for the interior nodes, which is performed in step 15, and assembling the RHS of the equation (6), which is performed in step 12. The most computationally demanding part of condensation is the factorisation of the inner-element matrix $\underline{\boldsymbol{A}}_{ii}^{-1}$. It is known [60] that the standard factorisation (including LU decomposition) algorithms have a cost $\mathcal{O}(n^3)$. Considering that the size of $\underline{\boldsymbol{A}}_{ii}$ can be described with equations (7) and (9), the resulting cost of factorising this matrix is $N_{el}N_{eq}^3(P-1)^9$ in 3D and $N_{el}N_{eq}^3(P-1)^6$ in 2D.

The second important operation is the Sparse Matrix-Matrix multiplications (SpGEMM). In our computations we rely on PETSc libraries to perform SpGEMM on compressed sparse row matrices. An upper bound for the cost of for matrix-matrix SpGEMM can be easily calculated assuming n matrix-vector SpMV. If the sparse matrix has nnz non-zero entries, then the matrix-matrix cost scales as $\mathcal{O}(n \times nnz)$. This estimate is not optimal and improved algorithms can be found in the literature [61, 62, 63], but this upper bound is accurate enough to analyse our condensed costs. To compute the condensed system, we perform two SpGEMM operations to compute $\underline{\boldsymbol{A}}_{bi}\underline{\boldsymbol{A}}_{ii}^{-1}\underline{\boldsymbol{A}}_{ib}$. We assume that $\underline{\boldsymbol{A}}_{ii}^{-1}$ has dense blocks of size $nb_{ii} = N_{eq}(P-1)^d$ and that the number of non-zeros is larger in $\underline{\boldsymbol{A}}_{ii}^{-1}$ than in the very sparse $\underline{\boldsymbol{A}}_{ib}$ (see Appendix C.27 for the estimation of the number of non-zeros in off-diagonal blocks of the Jacobian matrix, which scales as $N_{eq}^2(P+1)^2(4P+1)$). Taking into ac-

count that the size of the blocks of the Schur complement is $nb_{bb} = N_{eq}(6P^2 + 2)$ in 3D and $nb_{bb} = N_{eq}4P$ in 2D, we approximate the cost of the SpGEMM operation as $\mathcal{O}(N_{el}N_{eq}^3(6P^2+2)(P-1)^6)$ in 3D and $\mathcal{O}(N_{el}N_{eq}^3 4P(P-1)^4)$ in 2D. These upper bounds for matrix-matrix SpGEMM show that the inversion of the matrix $\underline{\boldsymbol{A}}_{ii}^{-1}$, which scales as $\mathcal{O}((P-1)^9)$ in 3D and as $\mathcal{O}((P-1)^6)$ in 2D is the dominant cost in calculating the Schur complement and obtaining the condensed system.

Finally, let us note that the estimation for nnz in Appendix C provides an upper bound that assumes full coupling between conservative variables. The real non-zero entries of $\underline{\boldsymbol{A}}_{bi}$ and $\underline{\boldsymbol{A}}_{ib}$ have few non-zeros, therefore in practical computations one would always expect a lower computational costs.

3.2. Cost of factorising the preconditioner

After computing the condensed system $\underline{\boldsymbol{A}}_{cond}$ in Algorithm 1 (step 8), we compute the preconditioner (step 9). As mentioned in section 2.4, we employ an element Block-Jacobi preconditioner to speed-up the convergence. If the full system (4) is considered, we factorise the whole element-blocks of matrix $\underline{\boldsymbol{A}}$ of size $N_{el}nb$, which has an operation count of $N_{el}N_{eq}^3(P+1)^9$ in 3D and $N_{el}N_{eq}^3(P+1)^6$ in 2D. If the condensed system is considered, we factorise the skeleton-element blocks of matrix $\underline{\boldsymbol{A}}_{cond}$ of size $N_{el}nb_{bb}$, which has a cost $N_{el}N_{eq}^3\left[(P+1)^d - (P-1)^d\right]^3$. This can be simplified to $N_{el}N_{eq}^3(6P^2+2)^3$ in 3D and $N_{el}N_{eq}^3(4P)^3$ in 2D. The cost of factorising the preconditioner is henceforth referred to as *preconditioner cost*. At this stage, we can al-

ready foresee that the cost of preconditioning the condensed system is much cheaper, since it scale as $\mathcal{O}(P^6)$ whilst for the full the cost scales as $\mathcal{O}(P^9)$.

Pardo et al. [24] concluded that their *hp*-FEM static condensation implementation for single, linear, second order PDE was computationally more efficient than the full system of equations when the number of iterations is high enough, since shorter times per iteration compensate the *condensation cost*. For time-dependent problems, like the compressible flow simulations considered here, this cost becomes even less important, as we can store the condensed matrix (in our matrix-based approach) and re-use it.

3.3. Cost of the preconditioned-GMRES solver

Step 13 in Algorithm 1 is detailed in Algorithm 2 where a preconditioned version of GMRES developed by Saad and Schultz [64] is presented. This is implemented in the PETSc library [49, 50, 51] and has been used in this work. In Algorithm 2, $\mathbf{R}$ and $\mathbf{V}$ represent the residual and its normalised version. m is dimension of the Krylov subspace $\underline{\boldsymbol{W}}_m$ with orthonormal vectors $\mathbf{W}_j$ and $\underline{\boldsymbol{H}}_m$ is the reduced Hessenberg matrix. $\underline{\boldsymbol{A}}$, $\Delta\mathbf{Q}$ and $\mathbf{B}$ represents either the full Jacobian matrix $\underline{\boldsymbol{A}}$, approximate solution $\Delta\boldsymbol{Q}$ and the right had side (RHS) $\boldsymbol{B}$ for the full system. Alternatively, when the condensed system is solved, we use the condensed Jacobian $\underline{\boldsymbol{A}}_{cond}$, $\Delta\boldsymbol{Q}_b$ and condensed RHS $\boldsymbol{B}_{cond}$.

Algorithm 2 Preconditioned GMRES-Solver

1: **function** GMRES-Solve($\Delta\mathbf{Q}, \underline{\boldsymbol{A}}, \underline{\boldsymbol{P}}^{-1}, \mathbf{B}$)
2: $\quad\mathbf{R}_0 \leftarrow \mathbf{B} - \underline{\boldsymbol{A}}\Delta\mathbf{Q}$
3: $\quad\mathbf{V}_1 \leftarrow \mathbf{R}_0/||\mathbf{R}_0||_2$
4: $\quad$**for** $j = 1, ..., m$ **do**
5: $\quad\quad\mathbf{Z}_j \leftarrow \underline{\boldsymbol{P}}^{-1}\mathbf{V}_j$
6: $\quad\quad\mathbf{W} \leftarrow \underline{\boldsymbol{A}}\mathbf{Z}_j$
7: $\quad\quad\underline{\boldsymbol{H}}_{i,j} \leftarrow \mathbf{W}^T\mathbf{V}_i,\ i = 1, ..., j$
8: $\quad\quad\mathbf{W} \leftarrow \mathbf{W} - \sum_{i=1}^{j} \underline{\boldsymbol{H}}_{i,j}\mathbf{V}_i$
9: $\quad\quad\mathbf{W} \leftarrow \underline{\boldsymbol{H}}_{j+1,j}/||\mathbf{W}||_2$
10: $\quad\quad\mathbf{V}_{j+1} \leftarrow \mathbf{W}/\underline{\boldsymbol{H}}_{j+1,j}$
11: $\quad\Delta\mathbf{Q} \leftarrow \Delta\mathbf{Q} + \underline{\boldsymbol{Z}}_m\mathbf{Y}_m$, where $\mathbf{Y}_m$ minimizes $||\beta\mathbf{e}_1 - \underline{\boldsymbol{H}}_m\mathbf{Y}||$
12: **return** $\Delta\mathbf{Q}$

The main costs within the GMRES iterative solver, arise from Sparse Matrix-Vector products (SpMV) (see steps 5 and 6 of Algorithm 2), which are governed by the number of non-zero entries nnz [65], in matrices $\underline{\boldsymbol{P}}^{-1}$ and $\underline{\boldsymbol{A}}$ [65]. Note that each nnz performs one multiplication and one addition, and we omit operation counts related to loading/storing variables. In addition to SpMV operations, GMRES also incorporates a large amount of purely vector operations (mainly dot products used to update the Hessenberg matrix, step in Algorithm 2). Their cost is proportional to the matrix size n, and have typically lower cost than sparse matrix-vector products. Therefore we focus only on SpMV operations.

The cost of Jacobian-SpMV (Step 6) is a function of nnz_{full} for full system $\underline{\boldsymbol{A}}$ and nnz_{cond} for the condensed system $\underline{\boldsymbol{A}}_{cond}$. In Appendix C, we have detailed the derivation of an upper bound for the number of non-zero entries for the Jacobian DGSEM matrix, see table 1 and Appendix C. Similarly, we

also express the number of non-zero entries in the condensed matrix nnz_{cond}, see equation (C.30) in Appendix C. This enables the calculation the costs of step 6: Precondition-SpMV $\mathbf{Z} = \underline{\boldsymbol{P}}^{-1}\mathbf{V}$ and Jacobian-SpMV $\mathbf{W} = \underline{\boldsymbol{A}}\mathbf{Z}$ in terms of (P, N_{eq}, d), as summarised in Table 2. Since the preconditioner is a locally dense matrix (block diagonal part is dense, while the off-diagonal parts are empty), we can bound the number of non-zero entries by the number of total entries in the diagonal blocks $nnz = N_{el}nb^2$. Therefore, the cost of the preconditioner-SpMV $\mathbf{Z} = \underline{\boldsymbol{P}}^{-1}\mathbf{V}$, presented in step 5 in Algorithm 2 can be expressed as $N_{el}N_{eq}^2(P+1)^6$ in 3D and $N_{el}N_{eq}^2(P+1)^4$ in 2D, if the full system is considered. For the condensed system, the costs are $N_{el}N_{eq}^2(6P^2+2)^2$ and $N_{el}N_{eq}^2 16P^2$ for 3D and 2D, respectively. The main preconditioned-GMRES costs are included in Table 2.

These estimates show asymptotic advantages for the condensed system, as P increases, for the two main steps within the preconditioned GMRES solver, further discussion can be found in the next section. In Section 4, Figures 2a and 6a report measured computational costs of GMRES in detail for the range of polynomial orders $P = 2, ..., 8$. The cost of GMRES (step 13 in Algorithm 1) is referred to as *solver cost*, in the following sections.

3.4. Summary of computational costs

Table 2 presents a summary of the estimated costs for the essential operations considered in the time stepping algorithm Algorithm 1, including the preconditioned-GMRES main steps. The biggest computational effort relates

to the factorisation of element-blocks needed to factorise the preconditioner for the full system and inner-element matrix $\underline{\boldsymbol{A}}_{ii}$ for the Schur complement, both scaling as $\mathcal{O}(P^9)$. As shown in the Table 2, factorising the blocks for the condensed preconditioner has a significant lower cost $\mathcal{O}(P^6)$. Similarly, the main GMRES steps favor from the use of static condensation. In 3D, both steps scale as $\mathcal{O}(P^4)$ for the condensed system, whilst they scale as $\mathcal{O}(P^5)$ and $\mathcal{O}(P^6)$ for the full system. These advantages are also expected in 2D simulations.

Table 2: Summary of the estimated leading costs of main operations in Algorithm 1 for 2D and 3D. Full and condensed systems are included.

3D		
	Full system	Condensed system
$\underline{\boldsymbol{A}}_{ii}^{-1}$	-	$N_{el}N_{eq}^3(P-1)^9$
SpGEMM	-	$N_{el}N_{eq}^3P^8$
$\underline{\boldsymbol{P}}^{-1}$	$N_{el}N_{eq}^3(P+1)^9$	$N_{el}N_{eq}^3P^6$
GMRES $\underline{\boldsymbol{A}}\mathbf{z}$	$N_{el}N_{eq}^2P^5$	$25N_{el}N_{eq}^2P^4$
GMRES $\underline{\boldsymbol{P}}^{-1}\mathbf{v}$	$N_{el}N_{eq}^2(P+1)^6$	$N_{el}N_{eq}^26P^4$
2D		
	Full system	Condensed system
$\underline{\boldsymbol{A}}_{ii}^{-1}$	-	$N_{el}N_{eq}^3(P-1)^6$
SpGEMM	-	$N_{el}N_{eq}^3P^5$
$\underline{\boldsymbol{P}}^{-1}$	$N_{el}N_{eq}^3(P+1)^6$	$N_{el}N_{eq}^364P^3$
GMRES $\underline{\boldsymbol{A}}\mathbf{z}$	$N_{el}N_{eq}^2P^4$	$13N_{el}N_{eq}^2P^2$
GMRES $\underline{\boldsymbol{P}}^{-1}\mathbf{v}$	$N_{el}N_{eq}^2(P+1)^4$	$N_{el}N_{eq}^216P^2$

In Section 4, we study the difference in computational costs for both, full and condensed Block-Jacobi preconditioners. There simulated costs are compared to the summarised estimated. We present the results in Figures 3a and 7a together with the *condensation costs*. Finally, we note that the use of

block preconditioners, that exploit the structure of DGSEM, has proven to be an important part in obtaining faster convergence rates for DG based solvers [33, 31, 32, 35, 36]. It has been advocated that Block-Jacobi preconditioner do not scale well in DG, which is indeed the case for the full system, since the block size scales with $(P+1)^3$, and associated cost $\mathcal{O}(P^9)$. However, the static condensed block size scales with $6P^2+2$ with costs $\mathcal{O}(P^6)$ in 3D and with $4P$ and cost $\mathcal{O}(P^3)$ in 2D, which renders Block-Jacobi preconditioner an interesting scalable preconditioner for the condensed GL-DGSEM approach.

4. Numerical results

We consider two test cases: a 3D manufactured solution and a 2D flow over NACA0012 airfoil at a high Angle of Attack (AOA) leading to an unsteady regime. The manufactured solution case illustrates the use of implicit time-marching solvers to reach a steady state solution, whilst the NACA0012 test case quantifies the improved cost in an unsteady flow simulation, with vortex shedding. The Mach number is set to Ma=0.1 for manufactured solution problem (other Ma and Re can be found in Appendix A) and Ma=0.3 for the NACA cases. For all the steady cases, we fix the final residual of the simulations to $||\underline{\boldsymbol{M}}^{-1}\boldsymbol{F}(\boldsymbol{Q})-\boldsymbol{S}||_\infty = 10^{-8}$ (see Algorithm 1) such that we compare the various schemes for the same accuracy.

The objective of the test cases is to validate the theoretical findings presented in the previous section. Therefore the main costs of the time marching scheme (see Algorithm 1) are compared with the theoretical cost estimations

(summarised in Table 2) for the two test cases. Additionally, the total cost to perform the simulations is included, to quantify the overall efficiency of the implicit statically condensed system compared to the full system.

4.1. Steady simulation: Manufactured Solution

The manufactured solution case is obtained by selecting an exact solution to the compressible Navier-Stokes equations:

$$\begin{aligned} &\rho = p = e^{-5\cdot(4(x-\frac{1}{2})^2+(y-\frac{1}{2})^2+(z-\frac{1}{2})^2)} + 1, \\ &u = v = w = 1, \end{aligned} \tag{11}$$

to then extract the balancing source terms:

$$\mathrm{s} = \begin{bmatrix} s_\rho \\ s_{\rho u} \\ s_{\rho v} \\ s_{\rho w} \\ s_{\rho E} \end{bmatrix} = \begin{bmatrix} 40(x-\frac{1}{2}) + 10(y-\frac{1}{2}) + 10(z-\frac{1}{2})) \\ 80(x-\frac{1}{2}) + 10(y-\frac{1}{2}) + 10(z-\frac{1}{2})) \\ 40(x-\frac{1}{2}) + 20(y-\frac{1}{2}) + 10(z-\frac{1}{2})) \\ 40(x-\frac{1}{2}) + 10(y-\frac{1}{2}) + 20(z-\frac{1}{2})) \\ \left[40(x-\frac{1}{2}) + 10(y-\frac{1}{2}) + 10(z-\frac{1}{2})\right]\left[\frac{5}{2} + \frac{1}{\gamma-1}\right] \end{bmatrix} \cdot \\ \cdot\ e^{-5(4(x-\frac{1}{2})^2+(y-\frac{1}{2})^2+(z-\frac{1}{2})^2)}. \tag{12}$$

We select the computational domain to be a $[0,1]^3$ cube with 64 hexahedral uniform elements. The solutions to the compressible Navier-Stokes equations (1) with the source term (12) can be seen in the Figure 1. Neither

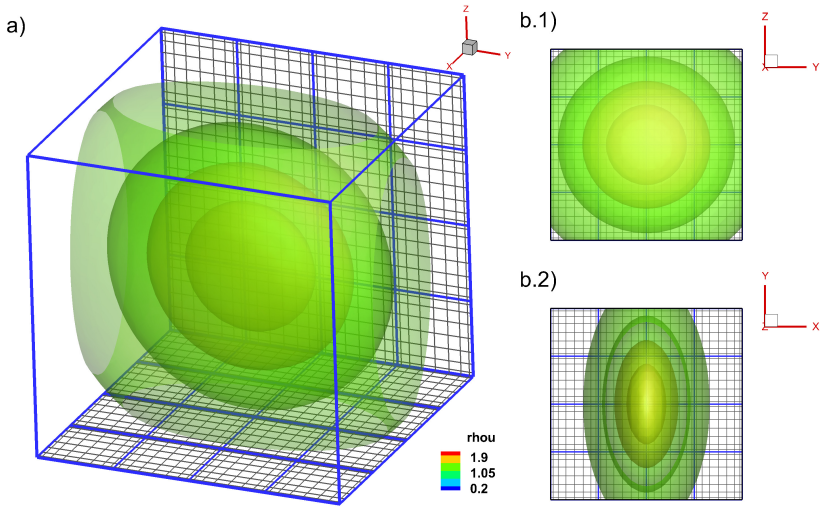


Figure 1: 3D Manufactured Solution: Solution of x-momentum ρu for a mesh of 64 hexahedral and polynomial order P=8. Figure a) 3D view, Figure b.1) and b.2) show cross-sections of yz and xy planes, respectively.

Mach nor Reynolds numbers have an impact on the final solution, however, both those parameters have a strong influence on the numerical scheme. We set the Reynolds number to Re = 1000 and the Mach to Ma = 0.1, but results for other Re and Ma can be found in Appendix B. The time-step size in the implicit computations is set $\Delta t = 0.1$ for all the polynomial orders and both systems.

Table 3 presents a summary of the conducted simulations for the full and condensed systems. We include the number of Jacobian updates i_{Jac} (identical for both full and condensed systems), the averaged number of Newton iterations per one time step $\frac{i_{Newton}}{i_{\Delta t}}$, the averaged number of GMRES iterations per one Newton solve $\frac{i_{GMRES}}{i_{Newton}}$ along with number of non-zero entries in full nnz_{full} and condensed nnz_{cond} systems. We observe that the number of

Newton iterations per one time step $\frac{i_{Newton}}{i_{\Delta t}}$ is constant for all polynomials and almost identical for the full and condensed system, consequently with using the same number of Jacobian updates in the full and condensed systems. The averaged number of GMRES iterations per one Newton solve $\frac{i_{GMRES}}{i_{Newton}}$, increases when using higher polynomial orders, scales similarly for both full and condensed systems. We also observe that the number on non-zeros is larger for the condensed system. This is not the expected behaviour for high polynomials, but due to the tight coupled stencil of the condensed system, this can be expected for low polynomial orders.

Table 3: 3D Manufactured Solution: Number of Jacobian updates i_{Jac} (identical for both full and condensed systems), averaged number of Newton iterations per one time step $\frac{i_{Newton}}{i_{\Delta t}}$ and averaged number of GMRES iterations per one Newton solve $\frac{i_{GMRES}}{i_{Newton}}$ along with number of non-zero entries in full nnz_{full} and condensed nnz_{cond} systems. For all cases considered in the table number of time steps needed to reach the steady state is $i_{\Delta t} = 50$, for polynomial orders $P = 2, .., 8$.

		Full system		Condensed system		Nonzero entries	
P	i_{Jac}	$\frac{i_{Newton}}{i_{\Delta t}}$	$\frac{i_{GMRES}}{i_{Newton}}$	$\frac{i_{Newton}}{i_{\Delta t}}$	$\frac{i_{GMRES}}{i_{Newton}}$	nnz_{full}	nnz_{cond}
2	3	6.4	3.5	6.4	3.5	6.5×10^5	1.2×10^6
3	3	6.1	4.4	6.1	4.4	2.2×10^6	6.6×10^6
4	3	5.9	5.7	6.1	5.5	6.0×10^6	2.2×10^7
5	3	6.1	6.7	6.1	6.5	1.3×10^7	5.6×10^7
6	4	6.3	7.7	6.4	7.6	2.8×10^7	1.2×10^8
7	4	6.5	8.6	6.6	8.5	5.3×10^7	2.2×10^8
8	5	6.6	10.0	6.6	9.9	9.4×10^7	3.9×10^8

Although the averaged linear solver iteration count is the same for both systems, this can be interpreted as an advantage of using static condensation with the cheaper skeleton-element Block-Jacobi. The similar iteration count has been observed in the past [24, 66] for finite element formulations (and

moderate polynomials P). There, the authors argued that even if the condition number of condensed Jacobian scales much better with P, the spectral radius of the iteration matrix, with a good preconditioner, is very similar for both systems, leading to similar number of iterations. Coherently with the findings of the aforementioned publications, we find almost the same number of iterations for full and condensed systems, but the latter being cheaper due to its smaller size, see Figure 2b.

The table is completed with Figures 2 and 3, where the total GMRES cost, the averaged solver cost per one linear system solve, the timing of factorising the preconditioner and the total simulation cost are depicted for the full and condensed systems and for polynomial orders $P = 2, .., 8$. The figures include the slopes for the theoretical estimates found in previous sections. Figure 2a splits the solver costs into the two main preconditioned-GMRES solver steps: preconditioner-SpMV $T_{\underline{\boldsymbol{P}}^{-1}\mathbf{v}}$ and Jacobian-SpMV $T_{\underline{\boldsymbol{A}}\mathbf{z}}$. Note that the rest of the GMRES costs are negligible. As estimated in Section 3, $T_{\underline{\boldsymbol{A}}\mathbf{z}}$ is larger for the condensed system due to higher number of non-zeros nnz, however the preconditioner-SpMV $T_{\underline{\boldsymbol{P}}^{-1}\mathbf{v}}$ is much cheaper and compensates $T_{\underline{\boldsymbol{A}}\mathbf{z}}$, which results in faster overall iterations. Additionally, the advantage of using static condensation in terms of solver costs becomes more noticeable for high polynomial orders. In all cases, the theoretical estimates are in good agreement with the numerical results.

Figure 3 presents the factorisation costs of the preconditioner along with condensation cost and the total time of the simulation. The factorisation of

the preconditioner matches well the theoretical estimates (see Table 2) for high enough polynomial orders. Discrepancies at low orders are attributed to the relatively small 3D problem considered and the effect of boundary conditions. In any case, it can be seen that despite the cost of condensing the system, the solver cost benefits from the condensation (Figure 2b), leading to overall faster solves, which illustrates the beneficial effect of using a condensed system for the higher polynomial orders.

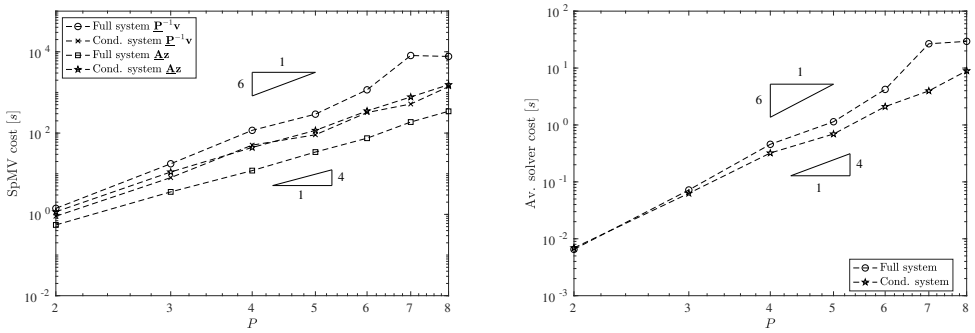
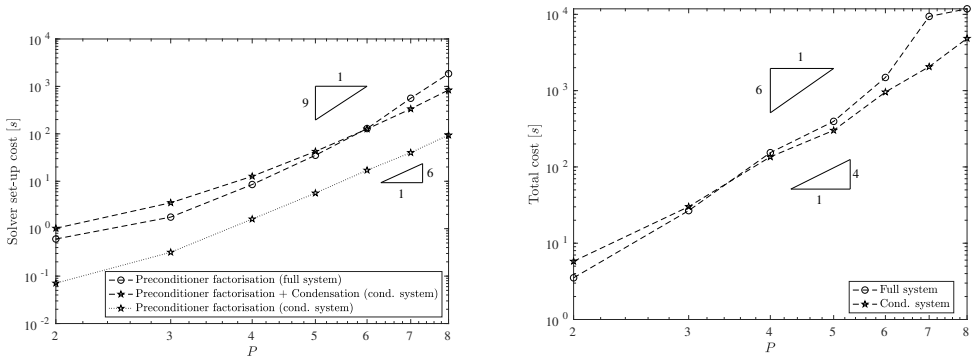


Figure 2: 3D Manufactured Solution: a) Total cost of the GMRES split in two major operations (in seconds) and b) Averaged GMRES solver cost (in seconds) per one linear system solve, for full and condensed systems for polynomial order $P = 2, .., 8$. Theoretical slopes are included depicted with a triangle.



(a) (b)

Figure 3: 3D Manufactured Solution: a) Timing of factorising the preconditioner (in seconds) and b) Total simulation cost (in seconds) to reach a tolerance 1×10^{-8} for the full and condensed systems for $P = 2, .., 8$. Theoretical slopes are included depicted with a triangle.

For completeness, we include a brief study for this problem, in Appendix B, for a range of Mach numbers, $0.1 \leq \text{Ma} \leq 0.8$ and Reynolds numbers, $200 \leq \text{Re} \leq 1000$ and show that the advantages of the static condensation are maintained for a wide range of flow conditions, and for a variety of polynomial orders.

4.2. Unsteady simulation: NACA0012 at AOA $= 20^o$

In this section, we challenge the static condensation technique for unsteady flows with application to aerodynamics. We simulate an unsteady NACA0012 case using a 2D computational squared domain of size 20×20 chords, with 1730 quadrilateral elements. Figure 4 depicts the h-mesh (in black) and the Gauss-Lobatto mesh (in gray) near the NACA0012 airfoil, and also the contours of x-momentum for the wake flow. To trigger vortex

shedding and study the performance of the implicit time-marching method for unsteady regimes, we set the Reynolds number to Re=200 and the angle of attack to AOA = 20^{o} (see Figure 4).

In steady problems, one of the main advantages of implicit time-integration schemes is that it is possible to increase the time-step size several orders of magnitude without losing accuracy or affecting stability [67]. However, in unsteady simulations the time-step size is bounded by accuracy constraints. This means that the time step in the implicit time-marching schemes has to be low enough to capture the physics of the problem, hence the performance of implicit time-marching schemes depends on the underlying physical problem at hand. In NACA0012, the characteristic physical time (one vortex shedding cycle) is 200 times larger than the time step selected for the implicit time-marching scheme. This restriction precludes the use of very large time steps in implicit solvers. For this reason, in the unsteady case, the implicit time step as been restricted to maintain accuracy (as shown in Figure 5). The time-step size in the explicit computations (ERK3), provided as reference for accuracy, is limited to $\Delta t = 2.0 \times 10^{-5}$, which is the maximum permitted by stability constraints for $P = 5$. In contrast, the time-step size in the implicit computations is set to $\Delta t = 1.0 \times 10^{-2}$, which is sufficiently low to capture the flow features accurately.

In this section, we show that the statically condensed DGSEM is able to outperform the standard full system for the same step size and that both methods provide accurate results. We provide results using an explicit RK3

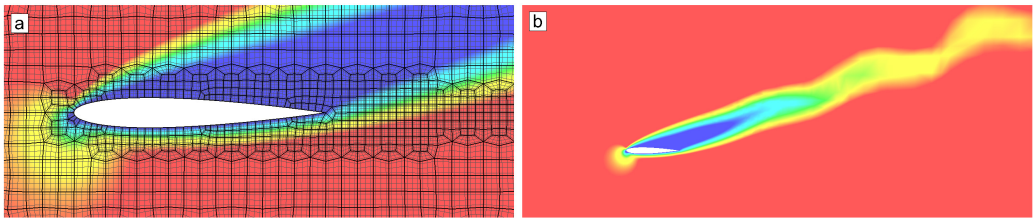


Figure 4: Unsteady 2D NACA0012: unsteady flow at Re=200 and AOA = 20°. Zoomed regions showing h-mesh (in black) and Gauss-Lobatto mesh (in gray) in a) and wake flow field in b). All figures include x-momentum contours.

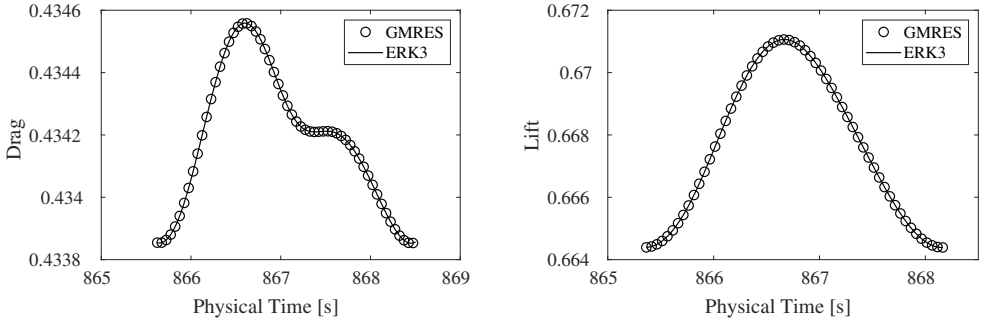
scheme as a reference. The comparison shows that the additional operations necessary to calculate the Schur complement, in the condensed system, do not damage the accuracy of the final solution with round-off errors.

Before proceeding with the costs, we present comparisons for the schemes in terms of accuracy, in Table 4. We simulate the unsteady flow for 10 vortex shedding cycles and compute mean lift, mean drag, and the Strouhal number. Let us note that once the polynomial order is fixed, the differences in mean lift, mean drag and Strouhal are negligible (i.e. below 10^{-5}) when using different time-marching schemes.

For completeness, Figure 5 depicts drag and lift curves for $P = 3$, computed with the explicit and implicit methods. We observe that explicit and implicit results match remarkably well, illustrating that there is no loss of accuracy when using implicit time-marching with moderate time steps.

Table 4: Unsteady 2D NACA0012: Time step, mean drag, mean lift and Strouhal number St; for explicit (ERK3) and implicit solver (GMRES) and polynomial orders $P = 2, 3, 4, 5$.

	$P=2$			$P=3$		
	ERK3	BDF2 full	BDF2 *cond.*	ERK3	BDF2 full	BDF2 *cond.*
Δt	2.7×10^{-5}	1.0×10^{-2}	1.0×10^{-2}	2.7×10^{-5}	1.0×10^{-2}	1.0×10^{-2}
Mean Drag	0.4383	0.4383	0.4383	0.4342	0.4342	0.4342
Mean Lift	0.6753	0.6753	0.6753	0.6677	0.6677	0.6677
St	0.3535	0.3530	0.3530	0.3565	0.3565	0.3565
	$P=4$			$P=5$		
	ERK3	BDF2 full	BDF2 *cond.*	ERK3	BDF2 full	BDF2 *cond.*
Δt	2.7×10^{-5}	1.0×10^{-2}	1.0×10^{-2}	2.7×10^{-5}	1.0×10^{-2}	1.0×10^{-2}
Mean Drag	0.4345	0.4345	0.4345	0.4342	0.4342	0.4342
Mean Lift	0.6664	0.6664	0.6664	0.6651	0.6651	0.6651
St	0.3577	0.3576	0.3576	0.3558	0.3558	0.3558



(a) (b)

Figure 5: Unsteady 2D NACA0012: Close-up comparison of explicit and implicit results for drag and lift for a single shedding cycle.

We now explore the different costs. Table 5 shows detailed information about number of Jacobian updates i_{Jac}, the averaged number of Newton iterations per one time step $\frac{i_{Newton}}{i_{\Delta t}}$ and averaged number of GMRES iterations per one Newton solve $\frac{i_{GMRES}}{i_{Newton}}$. As in the previous Manufactured Solution problem, conducting the simulation based on a smaller (but with more non-zeros) Jacobian matrix has almost no impact in the number of Newton iterations. Also like in the previous steady-state case, the averaged number of GMRES iterations is similar for both systems, but the iterations are, again, more efficient for the condensed system (Figure 6b). Unlike in the previous problem, the solver set-up costs (factorisation and condensation) do not constitute a big portion of the total simulation time, see Figure 7a, thus the advantage for the condensed system is clearly seen in Figure 7b. This is due to the fact that the Jacobian matrix is updated less frequently in this problem, and therefore the relative cost of the solver set-up in the total simulation cost is smaller. For this particular test case and range of polynomial orders, the solver set-up cost for the full system is cheaper than the theoretical prediction. However, it is still more costly than the condensation cost.

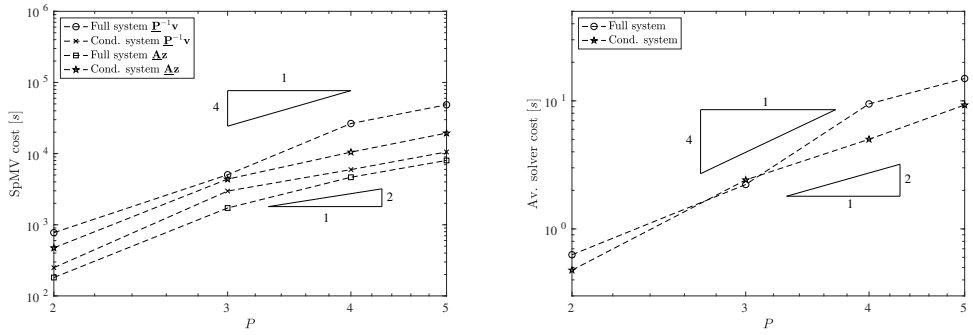
It can be seen that the static-condensation method provides the same accuracy up to given tolerance as the full system, but it is up to 40 % faster for the highest polynomial orders ($P = 4, 5$). As in the previous section, we also present the detailed results of the solver cost, Figures 6a and 6b. Again, the condensed system has more non-zeros nnz (Table 5), but the faster

preconditioner-SpMV compensates this cost and leads to faster simulations. Theoretical and measured preconditioner-SpMV operations for both systems agree well.

Finally, we can conclude that our static condensation time-marching method is more efficient for large polynomials, than the full system technique, even for unsteady problems, whilst providing accurate results.

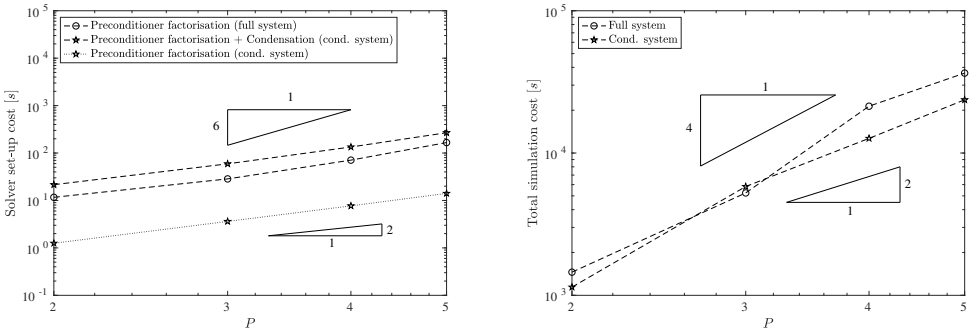
Table 5: Unsteady 2D NACA0012: Number of Jacobian updates i_{Jac} (computed only once and identical for both full and condensed systems), averaged number of Newton iterations per one time step $\frac{i_{Newton}}{i_{\Delta t}}$ and averaged number of GMRES iterations per one Newton solve $\frac{i_{GMRES}}{i_{Newton}}$ along with number of non-zero entries in full nnz_{full} and condensed nnz_{cond} systems. For all cases considered in the table number of time steps needed to compute one cycle is $i_{\Delta t} = 280$, for polynomial orders $P = 2, .., 5$.

		Full system		Condensed system		Nonzero entries	
P	i_{Jac}	$\frac{i_{Newton}}{i_{\Delta t}}$	$\frac{i_{GMRES}}{i_{Newton}}$	$\frac{i_{Newton}}{i_{\Delta t}}$	$\frac{i_{GMRES}}{i_{Newton}}$	nnz_{full}	nnz_{cond}
2	1	5.5	5.0	5.5	5.0	1.9×10^7	4.9×10^7
3		11.2	10.0	11.1	9.8	4.5×10^7	1.2×10^8
4		11.7	11.7	11.7	11.5	9.0×10^7	2.4×10^8
5		11.6	13.5	11.6	13.2	1.5×10^8	3.9×10^8



(a) (b)

Figure 6: Unsteady 2D NACA0012: a) Total cost of the GMRES split in two major operations (in seconds) and b) Averaged GMRES solver cost (in seconds) per one linear system solve, for the full and condensed systems for $P = 2, .., 5$. Theoretical slopes are included depicted with a triangle.



(a) (b)

Figure 7: Unsteady 2D NACA0012: a) Timing of factorising the preconditioner and b) Total simulation cost (in seconds) to complete one shedding period for full and condensed systems for $P = 2, .., 5$. Theoretical slopes are included depicted with a triangle.

5. Conclusion

We have analysed the advantages of performing static condensation on the compressible Navier-Stokes equations discretised with DGSEM and Gauss-

Lobatto points. The work focuses on the implicit preconditioned-GMRES time discretisations, and we have compared computational costs of solving the standard full Jacobian system to the static condensation technique for GL-DGSEM, detailed in Rueda-Ramírez et al. [1], both preconditioned using Block-Jacobi. To allow for fair comparisons, we split the costs into three categories: computation of the preconditioner, condensation costs for the statically condensed system and the solver GMRES cost to solve the full and condensed systems. We compare our numerical results with theoretical computational costs (Table 2), which include unpublished estimates for DGSEM. The theoretical estimates agree well with our simulations and provide solid bases for understanding the different costs involved.

For all cases included (steady-state 3D Manufactured Solution and unsteady 2D NACA0012), the static condensation shows accelerations (for large polynomial orders) due to the significantly faster solver time per single linear system solve. The accelerations are up to 30% for the Manufactured Solution and up to 40% for NACA0012 case, for the highest polynomial considered. Block-Jacobi preconditioner do not scale well with the polynomial order, which is indeed the case for the full system, since the element block Jacobian scales with $(P+1)^3$. However, we have shown that the statically condensed block size scales with $6P^2+2$ in 3D and with $4P$ in 2D, which renders Block-Jacobi preconditioner an interesting preconditioner for the condensed GL-DGSEM approach. Let us note that recent sum-factorisation techniques have been developed for high polynomials in discontinuous [48] and continu-

ous Galerkin [68] approaches hat decrease cost of factorising the blocks and show improved scalings for Block-Jacobi preconditioners. In the future, this approach may be applied to decrease the computational cost of condensed systems to further enhance the presented methodology. One drawback associated to the statically condensed system is the additional cost related to assembling the Schur complement (see Section 2.2 for more details). However, this cost is not high enough to mask the advantages of using static condensation, for high polynomial orders.

This manuscript compares iterative time-marching methods in serial, to avoid discrepancies due to parallelisation when comparing implicit techniques. Taking into account that Block-Jacobi preconditioners can be easily parallelised, we expect that future parallel implementation will lead to cheaper parallelised costs and less communication that when using the full system, as well as lower memory requirements. Future work, will assess the improvements in performance of implicit schemes (and especially of the static condensation methods) in many-core parallel environments and with more sophisticated preconditioners, including multilevel p-multigrid, specifically tailored for statically condensed systems.

Acknowledgements

Wojciech Laskowski and Esteban Ferrer would like to thank the European Union's Horizon 2020 Research and Innovation Program under the Marie Skłodowska-Curie grant agreement No 813605 for the ASIMIA ITN-

EID project. Additionally, Andrés Rueda-Ramírez acknowledges the funding received by the project SSeMID under the Marie Skłodowska-Curie grant agreement No 675008. Gonzalo Rubio acknowledges the funding received by the grant *Ayudas dirigidas al PDI para el fomento de la participación en solicitudes de proyectos H2020* from Universidad Politécnica de Madrid. Finally, the authors gratefully acknowledge the Universidad Politécnica de Madrid (www.upm.es) for providing computing resources on Magerit Supercomputer. Finally, the authors thank the reviewers for suggesting a deeper analysis by means of theoretical estimates, which have clearly improved the manuscript.

Appendix A. Preliminary assessment of preconditioners

In this section, we perform a preliminary study to assess the efficiency of several preconditioners: Block Jacobi and Incomplete LU (i.e. ILU(0), ILU(1) and ILU(2)). We compare the effect of the preconditioning and reordering in both the full and statically condensed systems.

For this preliminary selection, a manufactured with 8 hexahedral elements is selected. This case is smaller that the one considered in Section 4.1. We also increased the tolerance for the linear solver to $\text{TOL}_{GMRES} = e \cdot 0.3^i$ along with decreasing time step to $dt = 1e-2$ for more accurate results. The source term and rest of the parameters are maintained and can be found in Section 4.1.

Figure A.8.a, Figure A.8.b and Figure A.8.c show the average number of

iterations per Newton-Raphson step, the average solver cost and the cost of factorising the preconditioner. As can be seen in Figure A.8.a, even the simplest preconditioners considered (Block-Jacobi and ILU(0)) keep the average number of iterations low, even for high polynomial orders. The number of iterations remains unaltered by the use of static condensation. Figures A.8.b and A.8.c show the averaged solver cost and factorisation cost. Both of them present shorter times for the condensed system than for the full system. The difference between full and condensed systems, in the cost of factorisation for ILU(k), increases when increasing the filling k, as expected for more evolved preconditioners, but we note that the cost is lower for the condensed system, since the system size is smaller. Also the difference between the full and the condensed system in the average solver cost increases for ILU(k) for higher fillings k. Again, the condensed system cost is smaller.

From this preliminary analysis, we have chosen the Block-Jacobi for the rest of the paper. The reason is that, although non being optimal in terms of average iteration count, it presents a low memory cost, takes advantage of the element structure in DGSEM and can be easily parallelised, therefore, the results with element Block-Jacobi may provide better bases for further research.

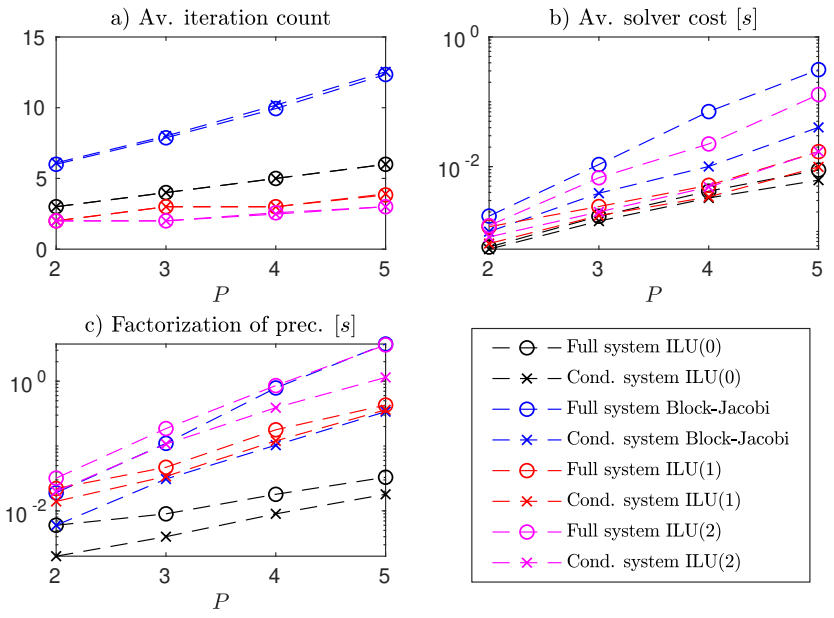


Figure A.8: a) Averaged iteration count (per linear system solve), b) averaged solver cost (per linear system solve) and c) factorisation cost of various types of preconditioners: Block-Jacobi, ILU(0), ILU(1), ILU(2).

Appendix B. Influence of Mach and Reynolds

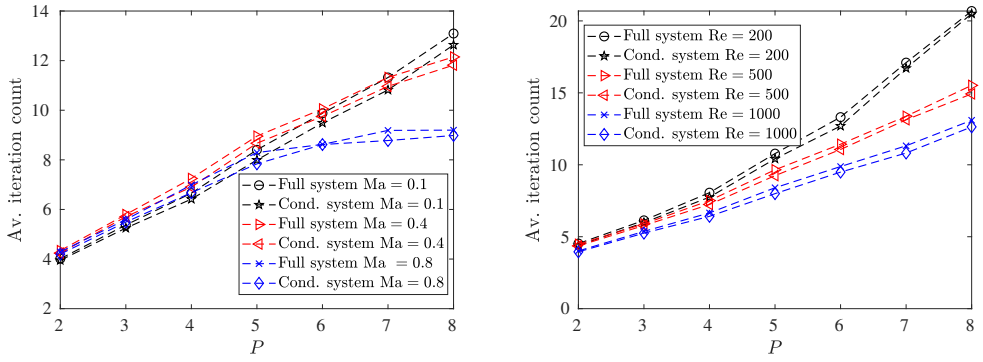


Figure B.9: Influence of a) Mach number (Ma) and b) Reynolds number (Re) on averaged number of iterations. All the different cases for Mach number were simulated with Re = 1000. The Reynolds study was conducted using Ma = 0.1.

In this section, we compare full and statically condensed systems for a range of Mach numbers, $0.1 \leq \text{Ma} \leq 0.8$ and Reynolds numbers, $200 \leq \text{Re} \leq 1000$. We use Block-Jacobi preconditioning for all cases.

Figure B.9 depicts iteration count for three different Mach numbers B.9a and three different Reynolds numbers B.9b. As expected, an increase in the Mach number (in the subsonic range) or in the Reynolds number, has a positive impact on the averaged iteration count for both condensed and full systems. We have not observed significant differences in computational efficiencies. In most cases, the static condensation provided very similar speed-up as depicted in the Figures 2 and 3. Overall, the static condensation system shows small improvements, over the full system, for this test case.

Appendix C. Estimation of non-zero entries in the Jacobian Matrix

The number of non-zero entries in the Jacobian matrix of a DGSEM discretisation depends on the nodes chosen (Gauss or Gauss-Lobatto), on the specifics of the flux (whether it has advection and/or diffusion terms), and on the surface numerical fluxes used. In this section, we derive the analytical expressions for the number of non-zero entries in the Gauss-Lobatto DGSEM Jacobian matrix for systems of nonlinear conservation laws with advection and diffusion terms, and surface numerical fluxes with compact support, as these are the subjects of the present study.

To facilitate the analysis, we will note the number of non-zero entries in a single diagonal block of the Jacobian matrix as nnz_d, and the number of non-zero entries in a single off-diagonal block as nnz_o. These expressions should be considered as an upper bound, since the number of non-zeros might be smaller due to the nonlinearities of the problem or the specific properties of the curvilinear mapping, as will be evident in next sections.

A system of nonlinear conservation laws reads

$$\partial_t \mathbf{q} + \vec{\nabla} \cdot \left(\overset{\leftrightarrow}{\mathbf{f}}^a - \overset{\leftrightarrow}{\mathbf{f}}^\nu \right) = \mathbf{0}, \tag{C.1}$$

where $\mathbf{q}$ is the state vector of conserved quantities, $\overset{\leftrightarrow}{\mathbf{f}}^a$ is the advective flux, and $\overset{\leftrightarrow}{\mathbf{f}}^\nu$ is the diffusive flux. Let us analyse the advection and diffusion terms separately.

Appendix C.1. Advection terms

Given a DGSEM discretisation of the advection terms in (C.1), an entry in the diagonal block that connects degrees freedom h and w of a specific element reads [44, 1]

$$\mathrm{DT}^a_{hw} = -\int_\Omega (\underline{\underline{\mathbf{J}}}^a\phi)_w \cdot \vec{\nabla}\phi_h \mathrm{d}\Omega + \oint_{\partial\Omega} \hat{\mathbf{f}}^a_{\mathbf{q}^+}\phi_w\phi_h \mathrm{d}S + \oint_{\partial\Omega\cap\Gamma} \hat{\mathbf{f}}^a_{\mathbf{q}^-}\mathbf{q}^-_{\mathbf{q}^+}\phi_w\phi_h \mathrm{d}S, \quad \text{(C.2)}$$

where Ω is the domain of the element where the degrees of freedom h and w are located, $\partial\Omega$ is the boundary of that domain, $\partial\Omega \cap \Gamma$ is the part of that boundary that belongs to a physical boundary, $\underline{\underline{\mathbf{J}}}^a = \partial\overleftrightarrow{\mathbf{f}}^a/\partial\mathbf{q}$ is the Jacobian of the advective flux, ϕ_w and ϕ_h are the basis functions that correspond to the degrees of freedom w and h, $\mathbf{q}^+$ and $\mathbf{q}^-$ are the inner and outer solutions on an element boundary, respectively, $\hat{\mathbf{f}}^a$ is the so-called (advective) surface numerical flux, $\hat{\mathbf{f}}^a_{\mathbf{q}^\pm}$ is its Jacobian with respect to the solution on the element boundary, and $\mathbf{q}^-_{\mathbf{q}^+}$ is the Jacobian of the boundary condition.

The first term of (C.2) generates the densest sparsity. This term can be rewritten using the contravariant fluxes [69] as

$$(\underline{\underline{\mathbf{J}}}^a\phi)_w \cdot \vec{\nabla}\phi_h = \underbrace{(\underline{\underline{\tilde{\mathbf{J}}}}^a\phi)_w \cdot \vec{\nabla}_\xi\phi_h}_{\text{Contravariant form}} = (\underline{\tilde{\boldsymbol{J}}}^a_1\phi)_w\frac{\partial\phi_h}{\partial\xi} + (\underline{\tilde{\boldsymbol{J}}}^a_2\phi)_w\frac{\partial\phi_h}{\partial\eta} + (\underline{\tilde{\boldsymbol{J}}}^a_3\phi)_w\frac{\partial\phi_h}{\partial\zeta}, \quad \text{(C.3)}$$

where $\vec{\xi} = (\xi, \eta, \zeta)$ are the coordinates on a reference element $\Omega_\xi = [-1, 1]^3$

that is mapped to physical space with high order polynomials

$$\Omega_\xi \xrightarrow{\vec{x}(\vec{\xi})} \Omega. \tag{C.4}$$

The degrees of freedom indexes h and w can be replaced by the tensor product coordinate indexes $h \leftarrow (i,j,k)$ and $w \leftarrow (r,s,t)$. This allows us to rewrite the basis functions as a tensor product combination of Lagrange interpolating polynomials,

$$\phi_h(\vec{x}(\vec{\xi})) = \ell_i^\xi(\xi)\ell_j^\eta(\eta)\ell_k^\zeta(\zeta) \tag{C.5}$$

$$\phi_w(\vec{x}(\vec{\xi})) = \ell_r^\xi(\xi)\ell_s^\eta(\eta)\ell_t^\zeta(\zeta). \tag{C.6}$$

As a result, (C.3) can be rewritten as

$$\begin{aligned}(\tilde{\underline{\underline{\mathbf{J}}}}^a\phi)_w \cdot \vec{\nabla}_\xi\phi_h =& (\underline{\tilde{\boldsymbol{J}}}_1^a)_{rst}\frac{\partial\ell_i^\xi}{\partial\xi}\ell_r^\xi\underbrace{\ell_s^\eta\ell_j^\eta}_{\delta_{sj}}\underbrace{\ell_t^\zeta\ell_k^\zeta}_{\delta_{tk}} \\ &+(\underline{\tilde{\boldsymbol{J}}}_2^a)_{rst}\frac{\partial\ell_j^\eta}{\partial\eta}\ell_s^\eta\underbrace{\ell_r^\xi\ell_i^\xi}_{\delta_{ri}}\underbrace{\ell_t^\zeta\ell_k^\zeta}_{\delta_{tk}} \\ &+(\underline{\tilde{\boldsymbol{J}}}_3^a)_{rst}\frac{\partial\ell_k^\zeta}{\partial\zeta}\ell_t^\zeta\underbrace{\ell_r^\xi\ell_i^\xi}_{\delta_{ri}}\underbrace{\ell_s^\eta\ell_j^\eta}_{\delta_{sj}},\end{aligned} \tag{C.7}$$

where δ is Dirac's delta function. Equation (C.7) only takes non-zero values if

$$(s = j \text{ and } t = k) \text{ or } (t = k \text{ and } t = k) \text{ or } (s = j \text{ and } r = i). \tag{C.8}$$

In consequence, there are connectivities between each degree of freedom $h \leftarrow (i, j, k)$ and all degrees of freedom $w \leftarrow (r, s, t)$ that lie along lines of the reference element coordinates. These connectivities appear as non-zero values in the Jacobian matrix, which leads to the following number of non-zeros for the diagonal blocks:

$$nnz_d^a\Big|_{2D} = N_{eq}^2(P+1)^2[2(P+1)-1]. \tag{C.9}$$

$$nnz_d^a\Big|_{3D} = N_{eq}^2(P+1)^3[3(P+1)-2]. \tag{C.10}$$

An entry in the off-diagonal block that connects the degrees of freedom h and w reads [44, 1]

$$\mathrm{ODT}_{hw}^a = \oint_{\partial\Omega\setminus\Gamma} \hat{\mathbf{f}}_{\mathbf{q}^-}^a \phi_w^- \phi_h \mathrm{d}S, \tag{C.11}$$

where ϕ_w^- is the basis function that corresponds to the degree of freedom w, which belongs to an element that is a neighbor of Ω across the interface $\partial\Omega \setminus \Gamma$.

It is evident that ODT_{hw}^a only takes non-zero values if h and w are both degrees of freedom on the boundary $\partial\Omega \setminus \Gamma$. As a result, the number of

non-zero entries for each off-diagonal block reads

$$nnz_o^a\Big|_{2D} = N_{eq}(P+1) \tag{C.12}$$

$$nnz_o^a\Big|_{3D} = N_{eq}(P+1)^2 \tag{C.13}$$

Appendix C.2. Diffusion terms

Neglecting the advective and time-dependent terms in (C.1), an entry in the diagonal block that connects degrees freedom h and w of a specific element reads [44, 1]

$$\begin{aligned}
\mathrm{DT}_{hw}^{\nu} = &\int_{\Omega} (\underline{\underline{\mathbf{J}}}^{\nu}\phi)_w \cdot \vec{\nabla}\phi_h \mathrm{d}\Omega \\
&+ \sum_{m=1}^{(P+1)^3} \left[\frac{1}{J_m \omega_m} \left(\int_{\Omega} \underline{\underline{\underline{\mathbf{G}}}}_m \phi_m \cdot \vec{\nabla}\phi_h \mathrm{d}\Omega \right) \cdot \left(- \int_{\Omega} \phi_w \vec{\nabla}\phi_m \mathrm{d}\Omega \right.\right. \\
&\quad \left.\left. + \oint_{\partial\Omega} \hat{\mathbf{q}}_{\mathbf{q}^+} \phi_w \phi_m \vec{n} \mathrm{d}S + \oint_{\partial\Omega\cap\Gamma} \hat{\mathbf{q}}_{\mathbf{q}^-} \mathbf{q}^-_{\mathbf{q}^+} \phi_w \phi_m \vec{n} \mathrm{d}S \right) \right] \\
&\quad - \oint_{\partial\Omega\setminus\Gamma} \left(\hat{\mathbf{f}}^{\nu}_{\mathbf{q}^+} \phi_w + \hat{\mathbf{f}}^{\nu}_{\vec{\nabla}\mathbf{q}^+} \cdot \vec{\nabla}\phi_w \right) \phi_h \mathrm{d}S \\
&\quad\quad - \oint_{\partial\Omega\cap\Gamma} \left(\frac{\partial \hat{\mathbf{f}}^{\nu}_{\Gamma}}{\partial \mathbf{q}^+} \phi_w + \frac{\partial \hat{\mathbf{f}}^{\nu}_{\Gamma}}{\partial \vec{\nabla}\mathbf{q}^+} \cdot \vec{\nabla}\phi_w \right) \phi_h \mathrm{d}S,
\end{aligned} \tag{C.14}$$

where $\underline{\underline{\mathbf{J}}}^{\nu} = \partial \overleftrightarrow{\mathbf{f}}^{\nu}/\partial \mathbf{q}$ is the Jacobian of the diffusive flux with respect to $\mathbf{q}$, J_m is the Jacobian of the mapping (C.4) at the node m, ω_m are the quadrature weights for the volume integral, $\underline{\underline{\underline{\mathbf{G}}}} = \partial \overleftrightarrow{\mathbf{f}}^{\nu}/\partial(\vec{\nabla}\mathbf{q})$ is the Jacobian of the diffusive flux with respect to $\vec{\nabla}\mathbf{q}$, $\hat{\mathbf{q}}$ is the numerical trace of the solution on the element boundary, $\hat{\mathbf{q}}_{\mathbf{q}^\pm}$ is the derivative of this numerical

trace with respect to the solutions on the element boundary, $\vec{n}$ is the outward-pointing normal vector on the boundary, $\hat{\mathbf{f}}^{\nu}_{\mathbf{q}^+}$ and $\hat{\mathbf{f}}^{\nu}_{\vec{\nabla}\mathbf{q}^+}$ are the Jacobians of the viscous surface numerical flux with respect to the solution and its gradient, respectively, and $\partial\hat{\mathbf{f}}^{\nu}_{\Gamma}/\partial\mathbf{q}^+$ and $\partial\hat{\mathbf{f}}^{\nu}_{\Gamma}/\partial(\vec{\nabla}\mathbf{q}^+)$ are the Jacobians of the viscous surface numerical flux on the physical boundaries. Note that the terms with the subscript Γ contain all the information of the boundary condition on the viscous surface numerical flux:

$$\frac{\partial\hat{\mathbf{f}}^{\nu}_{\Gamma}}{\partial\mathbf{q}^+} = \hat{\mathbf{f}}^{\nu}_{\mathbf{q}^+} + \hat{\mathbf{f}}^{\nu}_{\mathbf{q}^-}\mathbf{q}^-_{\mathbf{q}^+} + \hat{\mathbf{f}}^{\nu}_{\vec{\nabla}\mathbf{q}^-}(\vec{\nabla}\mathbf{q}^-)_{\mathbf{q}^+}, \text{ and} \tag{C.15}$$

$$\frac{\partial\hat{\mathbf{f}}^{\nu}_{\Gamma}}{\partial\vec{\nabla}\mathbf{q}^+} = \hat{\mathbf{f}}^{\nu}_{\vec{\nabla}\mathbf{q}^+} + \hat{\mathbf{f}}^{\nu}_{\vec{\nabla}\mathbf{q}^-}(\vec{\nabla}\mathbf{q}^-)_{\vec{\nabla}\mathbf{q}^+}, \tag{C.16}$$

The term first term of the summation in (C.14) is the one that generates the densest sparsity, as it is the multiplication of two volume integrals. This term can be expanded as

$$\begin{aligned}
\begin{pmatrix}\text{densest}\\ \text{term}\end{pmatrix} &= \sum_{m=1}^{(P+1)^3}\left[\frac{1}{J_m\omega_m}\left(\int_\Omega \underline{\underline{\mathbf{G}}}_m\phi_m\cdot\vec{\nabla}\phi_h\mathrm{d}\Omega\right)\cdot\left(-\int_\Omega\phi_w\vec{\nabla}\phi_m\mathrm{d}\Omega\right)\right]\\
&= -\sum_{m=1}^{(P+1)^3}\frac{1}{J_m\omega_m}\left[\left(\int_\Omega(\underline{\underline{\mathbf{G}}}_1\phi)_m\cdot\vec{\nabla}\phi_h\mathrm{d}\Omega\right)\left(\int_\Omega\phi_w\frac{\partial\phi_m}{\partial x}\mathrm{d}\Omega\right)\right.\\
&\qquad + \left(\int_\Omega(\underline{\underline{\mathbf{G}}}_2\phi)_m\cdot\vec{\nabla}\phi_h\mathrm{d}\Omega\right)\left(\int_\Omega\phi_w\frac{\partial\phi_m}{\partial y}\mathrm{d}\Omega\right)\\
&\qquad \left. + \left(\int_\Omega(\underline{\underline{\mathbf{G}}}_3\phi)_m\cdot\vec{\nabla}\phi_h\mathrm{d}\Omega\right)\left(\int_\Omega\phi_w\frac{\partial\phi_m}{\partial z}\mathrm{d}\Omega\right)\right].
\end{aligned} \tag{C.17}$$

The volume integrals on the left, that depend on the third-order tensors $\underline{\underline{\mathbf{G}}}_m$, imply two-point connectivities (as in (C.8)) for the degrees of freedom m and h. The volume integrals on the right imply two-point connectivities for the degrees of freedom w and m. In consequence, each degree of freedom $h \leftarrow (i,j,k)$ is connected with non-zeros with all degrees of freedom $w \leftarrow (r,s,t)$ that lie on the same $\xi-\eta$, $\eta-\zeta$ and $\xi-\zeta$ planes of reference element coordinates. Hence, the number of non-zero entries for in the Jacobian matrix in each diagonal block is

$$nnz_d^{\nu}\Big|_{2D} = N_{eq}^2(P+1)^4 \tag{C.18}$$

$$nnz_d^{\nu}\Big|_{3D} = 3N_{eq}^2P(P+1)^4. \tag{C.19}$$

It is important to point out that the sparsity pattern generated by (C.17) contains all the non-zero entries needed for the other diffusive terms and for the advective terms. As can be seen, the diffusive terms generate dense diagonal blocks in 2D.

An entry in the off-diagonal block that connects the degrees of freedom h and w reads [1]

$$\mathrm{ODT}_{hw}^{\nu} = \sum_{m=1}^{(P+1)^3} \left[\frac{1}{J_m\omega_m} \left(\int_{\Omega} \underline{\underline{\mathbf{G}}}_m \phi_m \cdot \vec{\nabla}\phi_h \mathrm{d}\Omega \right) \cdot \left(\oint_{\partial\Omega\setminus\Gamma} \phi_w^- \phi_m \vec{n} \mathrm{d}S \right) \right] - \oint_{\partial\Omega\setminus\Gamma} \left(\hat{\mathbf{f}}_{\mathbf{q}^-}^{\nu} \phi_w^- + \hat{\mathbf{f}}_{\vec{\nabla}\mathbf{q}^-}^{\nu} \vec{\nabla}\phi_w^- \right) \phi_h \mathrm{d}S. \tag{C.20}$$

714 In this case, both the summation term and the single surface integral of
715 (C.20) play an important role in the sparsity of the off-diagonal blocks.

Let us analyse the summation term first. The volume integral implies two-point connectivities for the degrees of freedom m and h, and the surface integral only takes non-zero values if the degrees of freedom w and m lie on an element boundary. As a result, each degree of freedom $h \leftarrow (i, j, k)$ is connected with non-zeros with the degree of freedom (of a neighbor element) $w \leftarrow (r, s, t)$ that lies on the element boundary *and* on the same iso-ξ_i line as h. Therefore, the number of non-zeros due to the summation term is

$$nnz_o^{\nu}\Big|_{1,2D} = N_{eq}^2(P+1)^2. \tag{C.21}$$

$$nnz_o^{\nu}\Big|_{1,3D} = N_{eq}^2(P+1)^3. \tag{C.22}$$

The single surface integral in (C.20) is important for the sparsity pattern since it contains the gradient of the basis functions on the neighbor element, $\vec{\nabla}\phi_w^-$. This term can be written explicitly as

$$\vec{\nabla}\phi_w^- = \begin{pmatrix} \frac{\partial \phi_w^-}{\partial x} \\ \frac{\partial \phi_w^-}{\partial y} \\ \frac{\partial \phi_w^-}{\partial z} \end{pmatrix} = \begin{pmatrix} \sum_{p=1}^{d} \frac{\partial \phi_w^-}{\partial \xi_p}\frac{\partial \xi_p}{\partial x} \\ \sum_{p=1}^{d} \frac{\partial \phi_w^-}{\partial \xi_p}\frac{\partial \xi_p}{\partial y} \\ \sum_{p=1}^{d} \frac{\partial \phi_w^-}{\partial \xi_p}\frac{\partial \xi_p}{\partial z} \end{pmatrix}. \tag{C.23}$$

Note that the sparsity pattern that this term generates depends on the geometry mapping ($\partial\vec{\xi}/\partial\vec{x}$) and on the position of the degrees of freedom w and h. For a general curvilinear mapping, the second term of (C.20) is zero

when h is not a degree of freedom on the element boundary or when

$$\frac{\partial \phi_w^-}{\partial \xi} = \frac{\partial \phi_w^-}{\partial \eta} = \frac{\partial \phi_w^-}{\partial \zeta} = 0. \tag{C.24}$$

Therefore, for each h on the element boundary, there are non-zeros for the degrees of freedom w of a neighbor element that are arranged along lines of the reference coordinates. In summary, the number of non-zero entries for each off-diagonal block due to the second term of (C.20) is

$$nnz_o^\nu\Big|_{2,2D} = N_{eq}^2(P+1)[2(P+1)-1]. \tag{C.25}$$

$$nnz_o^\nu\Big|_{2,3D} = N_{eq}^2(P+1)^2[3(P+1)-2]. \tag{C.26}$$

Remark that the term that leads to the non-zero pattern (C.21) shares some non-zeros with the term that leads to (C.25). Combining (C.21) and (C.25), and accounting for the repeated non-zero entries, the total number of non-zeros in an off-diagonal block is

$$\begin{aligned} nnz_o^\nu\Big|_{2D} &= N_{eq}^2\left[P(P+1)+(P+1)[2(P+1)-1]\right] \\ &= N_{eq}^2(P+1)(3P+1). \end{aligned} \tag{C.27}$$

$$\begin{aligned} nnz_o^\nu\Big|_{3D} &= N_{eq}^2\left[P(P+1)^2+(P+1)^2[3(P+1)-2]\right] \\ &= N_{eq}^2(P+1)^2(4P+1). \end{aligned} \tag{C.28}$$

Appendix C.3. Total number of non-zero entries

The number of non-zero entries in the diagonal and off-diagonal blocks depends on the position of the element, i.e. both blocks are more dense for interior elements connected purely to other interior elements. In our calculations, we disregard the boundary elements and estimate the upper bound for the total number of non-zero entries in the Jacobian Matrix:

$$nnz_{full} = N_{el} nnz_d + (C_{Neigh} N_{el} - N_{Out}) nnz_o, \tag{C.29}$$

where C_{Neigh} is an upper bound of neighbouring elements ($C_{Neigh2D} = 4$ and $C_{Neigh3D} = 6$) and N_{Out} is total number of element faces (3D) or edges (2D) on the boundary of computational domain. For the cubic mesh used for Manufactured Solution problem $N_{OutMS3D} = 6(N_{el}^{\frac{1}{3}})^2 = 96$ and for the NACA0012 case $N_{OutNACA0012} = 880$. The accuracy of theses estimations can be found in Figure C.10. The theoretical curve overestimate the number of non-zero entries due to the assumptions that were undertaken to estimate non-zeros in each block and the fact that all the estimated blocks disregard physical boundary conditions (boundary blocks have significantly less non-zero entries). The slopes however, follow the same trend within considered range of polynomials. The reason for the undershoot is twofold. First, the Jacobian matrices for the Navier-Stokes equations ($\underline{\underline{\mathbf{G}}}$, $\underline{\underline{\mathbf{J}}}^a$ and $\underline{\underline{\mathbf{J}}}^\nu$) are far from dense (see [44, 1]). Second, the mesh for this case is Cartesian and

therefore $\partial \xi_i / \partial x_j = 0$ for $i \neq j$.

Now we estimate the number of non-zeros in the condensed system. Due to the two matrix-matrix products (see Section 3.1) needed to compute the Schur complement, the number of non-zero entries in the condensed system significantly increases. The non-zero entries in each block are constrained by the block size, which has complexity (10) ($nb_{bb} = N_{eq} 4P$ in 2D and $nb_{bb} = N_{eq}(6P^2 + 2)$ in 3D). However, the SpGEMM operations introduce new non-zero entries into the matrix $\underline{\boldsymbol{A}}_{cond}$. Additionally, the stencil of the block structure in the Schur complement is wider (non-compact) than in the Jacobian matrix. Therefore, the upper bound for the non-zero entries in the condensed system is

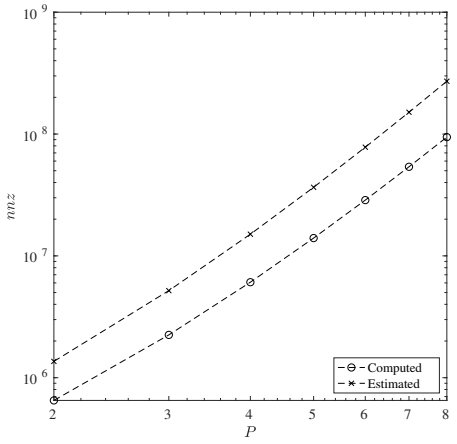
$$\text{3D: } nnz_{cond} = C_{NeighNeigh} N_{el} N_{eq}^2 (6P^2 + 2)^2, \tag{C.30}$$

$$\text{2D: } nnz_{cond} = C_{NeighNeigh} N_{el} N_{eq}^2 (4P)^2, \tag{C.31}$$

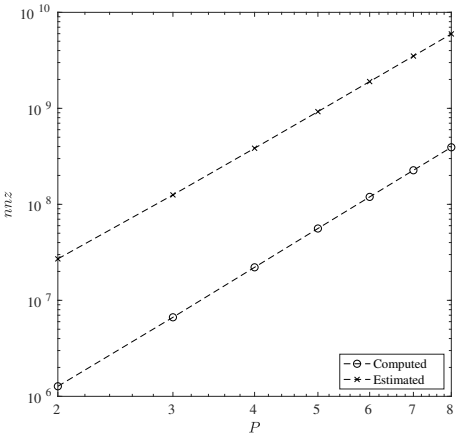
where the constants $C_{NeighNeigh3D} = 25$ and $C_{NeighNeigh2D} = 13$ place an upper bound on the total number of blocks per row in the condensed system. Note that these constants have been obtained based on the connectives of structured meshes and can be slightly bigger for particular unstructured meshes.

Finally, Figure C.10 compares the theoretical estimated number of non-zero entries nnz, for the full and the condensed systems, to the values ex-

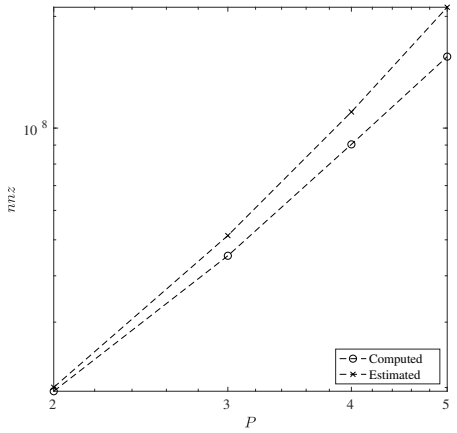
tracted from the simulations, using the 3D manufactured solution, see section 4.1, and the 2D NACA0012 airfoil, see section 4.2. The slopes agree well and it can be seen that the estimates over-predict the simulations in all cases, which follows for having derived upper bounds. Small slope discrepancies for the 2D cases can be explained as follows. Our 2D simulations are not truly 2D, but instead we have performed a 3D simulation with only one element in the third direction (and polynomial $P_z = 2$). An approximated upper bound for the nnz (and associated cost) for this particular situation has been obtained by assuming three two-dimensional simulations. For this reason when depicting the estimated value in Figure C.10, the estimate has been multiplied by a constant factor of three. This estimate does not properly account for boundary conditions, which explains small differences.



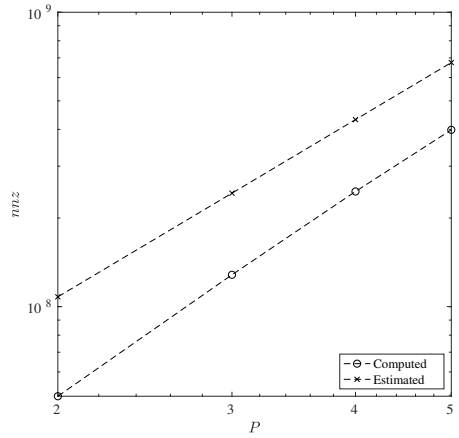
(a) Manufactured Solution Full System



(b) Manufactured Solution Condensed System



(c) NACA0012 Full System



(d) NACA0012 Condensed System

Figure C.10: Comparison of computed non-zero entries against estimations derived in (C.29), (C.31) and (C.30) for full and condensed systems of two cases considered in this work (Manufactured Solution and NACA0012).

References

[1] A. Rueda-Ramírez, E. Ferrer, D. Kopriva, G. Rubio, E. Valero, A statically condensed discontinuous Galerkin spectral element method on Gauss-Lobatto nodes for the compressible Navier-Stokes equations, 2019. arXiv:1911.02366.

[2] B. Cockburn, C.-W. Shu, The local discontinuous Galerkin method for time-dependent convection-diffusion systems, SIAM Journal on Numerical Analysis 35 (1998) 2440–2463.

[3] E. Ferrer and R.H.J. Willden, A high order discontinuous Galerkin finite element solver for the incompressible Navier–Stokes equations, Computers & Fluids 46 (2011) 224–230.

[4] E. Ferrer, R. H. Willden, A high order discontinuous Galerkin - Fourier incompressible 3D Navier-Stokes solver with rotating sliding meshes, Journal of Computational Physics 231 (2012) 7037–7056.

[5] E. Ferrer, An interior penalty stabilised incompressible discontinuous Galerkin–Fourier solver for implicit large eddy simulations, Journal of Computational Physics 348 (2017) 754–775.

[6] N. Fehn, M. Kronbichler, C. Lehrenfeld, G. Lube, P. W. Schroeder, High-order DG solvers for underresolved turbulent incompressible flows: A comparison of L2 and H(div) methods, International Journal for Numerical Methods in Fluids 91 (2019) 533–556.

[7] M. Kompenhans, G. Rubio, E. Ferrer, E. Valero, Adaptation strategies for high order discontinuous Galerkin methods based on Tau-estimation, Journal of Computational Physics 306 (2016) 216–236.

[8] M. Kompenhans, G. Rubio, E. Ferrer, E. Valero, Comparisons of p–adaptation strategies based on truncation– and discretisation–errors for high order discontinuous Galerkin methods, Computers & Fluids 139 (2016) 36 – 46. 13th {USNCCM} International Symposium of High-Order Methods for Computational Fluid Dynamics - A special issue dedicated to the 60th birthday of Professor David Kopriva.

[9] J. Manzanero, E. Ferrer, G. Rubio, E. Valero, Design of a Smagorinsky spectral Vanishing Viscosity turbulence model for discontinuous Galerkin methods, Computers & Fluids (2020) 104440.

[10] A. M. Rueda-Ramírez, J. Manzanero, E. Ferrer, G. Rubio, E. Valero, A p-multigrid strategy with anisotropic p-adaptation based on truncation errors for high-order discontinuous Galerkin methods, Journal of Computational Physics 378 (2019) 209–233.

[11] Z. J. Wang, K. Fidkowski, R. Abgrall, F. Bassi, D. Caraeni, A. Cary, H. Deconinck, R. Hartmann, K. Hillewaert, H. T. Huynh, et al., High-order CFD methods: current status and perspective, International Journal for Numerical Methods in Fluids 72 (2013) 811–845.

[12] K. Black, A conservative spectral element method for the approximation of compressible fluid flow, Kybernetika 35 (1999) 133–146.

[13] J. Manzanero, G. Rubio, E. Ferrer, E. Valero, D. A. Kopriva, Insights on aliasing driven instabilities for advection equations with application to Gauss–Lobatto discontinuous Galerkin methods, Journal of Scientific Computing 75 (2018) 1262–1281.

[14] G. J. Gassner, A. R. Winters, D. A. Kopriva, Split form nodal discontinuous Galerkin schemes with summation-by-parts property for the compressible Euler equations, Journal of Computational Physics 327 (2016) 39–66.

[15] A.R. Winters and G.J. Gassner, Affordable, entropy conserving and entropy stable flux functions for the ideal MHD equations, Journal of Computational Physics 304 (2016) 72 – 108.

[16] J. Manzanero, G. Rubio, D. A. Kopriva, E. Ferrer, E. Valero, A free–energy stable nodal discontinuous Galerkin approximation with summation–by–parts property for the Cahn–Hilliard equation, Journal of Computational Physics 403 (2020) 109072.

[17] J. Manzanero, G. Rubio, D. A. Kopriva, E. Ferrer, E. Valero, Entropy–stable discontinuous Galerkin approximation with summation–by–parts property for the incompressible Navier–Stokes/Cahn–Hilliard system, Journal of Computational Physics (2020) 109363.

[18] G. J. Gassner, A. R. Winters, F. J. Hindenlang, D. A. Kopriva, The BR1 scheme is stable for the compressible Navier–Stokes equations, Journal of Scientific Computing 77 (2018) 154–200.

[19] J. Manzanero, G. Rubio, D. A. Kopriva, E. Ferrer, E. Valero, An entropy–stable discontinuous Galerkin approximation for the incompressible Navier–Stokes equations with variable density and artificial compressibility, Journal of Computational Physics 408 (2020) 109241.

[20] G. Karniadakis and S.J. Sherwin, Spectral/hp Element Methods for Computational Fluid Dynamics, Oxford Scholarship, 2005.

[21] L. Haupt, J. Stiller, W. Nagel, A fast spectral element solver combining static condensation and multigrid techniques, Journal of Computational Physics 255 (2013) 384 – 395.

[22] E. Wilson, The static condensation algorithm, International Journal for Numerical Methods in Engineering 8 (1974) 198–203.

[23] I. Huismann, J. Stiller, J. Fröhlich, Scaling to the stars – a linearly scaling elliptic solver for p-multigrid, Journal of Computational Physics 398 (2019) 108868.

[24] D. Pardo, J. Álvarez Aramberri, M. Paszynski, L. Dalcin, V. Calo, Impact of element-level static condensation on iterative solver performance, Computers and Mathematics with Applications 70 (2015) 2331–2341.

[25] S. J. Sherwin, R. M. Kirby, J. Peiró, R. L. Taylor, O. C. Zienkiewicz, On 2D elliptic discontinuous Galerkin methods, International Journal for Numerical Methods in Engineering 65 (2006) 752–784.

[26] B. Cockburn, J. Gopalakrishnan, R. Lazarov, Unified hybridization of discontinuous Galerkin, mixed, and continuous Galerkin methods for second order elliptic problems, SIAM J. Numer. Anal 47 (2009) 1319–1365. doi:10.1137/070706616.

[27] J. Carrero, B. Cockburn, D. Schoetzau, Hybridized globally divergence-free LDG methods. part I: The Stokes problem, Math. Comput. 75 (2006) 533–563. doi:10.1090/S0025-5718-05-01804-1.

[28] M. Franciolini, K. Fidkowski, A. Crivellini, Efficient discontinuous Galerkin implementations and preconditioners for implicit unsteady compressible flow simulations, arXiv preprint (2018). arXiv:physics.comp-ph/1812.04789.

[29] J. Peraire, N. C. Nguyen, B. Cockburn, An embedded discontinuous Galerkin method for the compressible Euler and Navier-Stokes equations, 20th AIAA Computational Fluid Dynamics Conference 2011 (2011). doi:10.2514/6.2011-3228.

[30] K. J. Fidkowski, T. A. Oliver, J. Lu, D. L. Darmofal, p-Multigrid solution of high-order discontinuous Galerkin discretizations of the com-

pressible Navier-Stokes equations, Journal of Computational Physics 207 (2005) 92–113. doi:10.1016/j.jcp.2005.01.005.

[31] P. O. Persson, An efficient low memory implicit DG algorithm for time dependent problems, Collection of Technical Papers - 44th AIAA Aerospace Sciences Meeting 2 (2006) 1421–1431. doi:10.2514/6.2006-113.

[32] P. O. Persson, J. Peraire, Newton-GMRES preconditioning for discontinuous Galerkin discretizations of the Navier–Stokes equations, SIAM Journal on Scientific Computing 30 (2008) 2709–2733. doi:10.1137/070692108.

[33] L. T. Diosady, D. L. Darmofal, Preconditioning methods for discontinuous Galerkin solutions of the Navier-Stokes equations, Journal of Computational Physics 228 (2009) 3917–3935. doi:10.1016/j.jcp.2009.02.035.

[34] K. Shahbazi, D. J. Mavriplis, N. K. Burgess, Multigrid algorithms for high-order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations, Journal of Computational Physics 228 (2009) 7917–7940. doi:10.1016/j.jcp.2009.07.013.

[35] P. O. Persson, A sparse and high-order accurate line-based discontinuous Galerkin method for unstructured meshes, Journal of Computational Physics 233 (2013) 414–429. doi:10.1016/j.jcp.2012.09.008. arXiv:1204.1533.

[36] W. Pazner, P. O. Persson, Stage-parallel fully implicit Runge–Kutta solvers for discontinuous Galerkin fluid simulations, Journal of Computational Physics 335 (2017) 700–717. doi:10.1016/j.jcp.2017.01.050. arXiv:1701.07181.

[37] M. Franciolini, L. Botti, A. Colombo, A. Crivellini, p-Multigrid matrix-free discontinuous Galerkin solution strategies for the under-resolved simulation of incompressible turbulent flows, 2018. arXiv:1809.00866.

[38] P. Bastian, E. H. Müller, S. Muthing, M. Piatkowski, Matrix-free multigrid block-preconditioners for higher order discontinuous Galerkin discretisations, Journal of Computational Physics 394 (2019) 417 – 439. doi:https://doi.org/10.1016/j.jcp.2019.06.001.

[39] M. Franciolini, S. M. Murman, Multigrid preconditioning for a space-time spectral-element discontinuous-galerkin solver, AIAA Scitech 2020 Forum (2020). doi:10.2514/6.2020-1314.

[40] A. Pueyo, D. Zingg, An efficient Newton-GMRES solver for aerodynamic computations, 13th Computational Fluid Dynamics Conference (1997) 712–721. doi:10.2514/6.1997-1955.

[41] W. Anderson, R. D. Rausch, D. L. Bonhaus, Implicit/multigrid algorithms for incompressible turbulent flows on unstructured grids, Journal of Computational Physics 128 (1996) 391 – 408. doi:https://doi.org/10.1006/jcph.1996.0219.

[42] J. Gopalakrishnan, G. Kanschat, A multilevel discontinuous Galerkin method, Numer. Math. 95 (2003) 527–550. doi:10.1007/s002110200392.

[43] K. Black, A conservative spectral element method for the approximation of compressible fluid flow, Kybernetika 35 (1999) 133–146.

[44] A. M. Rueda-Ramírez, Efficient Space and Time Solution Techniques for High-Order Discontinuous Galerkin Discretizations of the 3D Compressible Navier-Stokes Equations, Ph.D. thesis, Universidad Politécnica de Madrid, 2019.

[45] A. Huerta, A. Angeloski, X. Roca, J. Peraire, Efficiency of high-order elements for continuous and discontinuous galerkin methods, International Journal for Numerical Methods in Engineering 96 (2013) 529–560. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.4547. doi:10.1002/nme.4547. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.4

[46] W. Habchi, Model Order Reduction (MOR) Techniques, 2018, pp. 297–338. doi:10.1002/9781119225133.ch8.

[47] P. Bastian, E. H. Müller, S. Müthing, M. Piatkowski, Matrix-free multigrid block-preconditioners for higher order discontinuous Galerkin discretisations, Journal of Computational Physics 394 (2019) 417 – 439.

[48] W. Pazner, P. O. Persson, Approximate tensor-product preconditioners for very high order discontinuous Galerkin methods, Journal of Com-

putational Physics 354 (2018) 344–369. doi:10.1016/j.jcp.2017.10.030. arXiv:1704.04549.

[49] S. Balay, W. D. Gropp, L. C. McInnes, B. F. Smith, Efficient management of parallelism in object oriented numerical software libraries, in: E. Arge, A. M. Bruaset, H. P. Langtangen (Eds.), Modern Software Tools in Scientific Computing, Birkhäuser Press, 1997, pp. 163–202.

[50] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, PETSc Users Manual, Technical Report ANL-95/11 - Revision 3.12, Argonne National Laboratory, 2019. URL: https://www.mcs.anl.gov/petsc.

[51] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, PETSc Web page, https://www.mcs.anl.gov/petsc, 2019. URL: https://www.mcs.anl.gov/petsc.

[52] F. Bassi, A. Crivellini, D. A. Di Pietro, S. Rebay, An implicit high-order discontinuous Galerkin method for steady and unsteady

incompressible flows, Computers and Fluids 36 (2007) 1529–1546. doi:10.1016/j.compfluid.2007.03.012.

[53] B. R. Ahrabi, D. J. Mavriplis, An implicit block ILU smoother for preconditioning of Newton–Krylov solvers with application in high-order stabilized finite-element methods, Computer Methods in Applied Mechanics and Engineering 358 (2020) 112637.

[54] J. Williamson, Low-storage Runge-Kutta schemes, Journal of Computational Physics 35 (1980) 48 – 56. doi:https://doi.org/10.1016/0021-9991(80)90033-9.

[55] R. Biswas, K. Devine, J. Flaherty, Parallel, adaptive finite element methods for conservation laws, Applied Numerical Mathematics 14 (1994) 255–283.

[56] N. Chalmers, G. Agbaglah, M. Chrust, C. Mavriplis, A parallel hp-adaptive high order discontinuous Galerkin method for the incompressible Navier-Stokes equations, Journal of Computational Physics: X 2 (2019) 100023. doi:https://doi.org/10.1016/j.jcpx.2019.100023.

[57] P. Birken, G. Gassner, M. Haas, C. D. Munz, Efficient time integration for discontinuous Galerkin method for the unsteady 3D Navier-Stokes equations, ECCOMAS 2012 - European Congress on Computational Methods in Applied Sciences and Engineering, e-Book Full Papers (2012) 4334–4353.

[58] M. J. Zahr, P.-O. Persson, Performance tuning of newton-gmres methods for discontinuous galerkin discretizations of the navier-stokes equations, in: 21st AIAA Computational Fluid Dynamics Conference, 2013, p. 2685.

[59] C. R. Nastase, D. J. Mavriplis, High-order discontinuous Galerkin methods using an hp-multigrid approach, Journal of Computational Physics 213 (2006) 330–357. doi:10.1016/j.jcp.2005.08.022.

[60] G. H. Golub, C. F. Van Loan, Matrix Computations, third ed., The Johns Hopkins University Press, 1996.

[61] F. G. Gustavson, Two fast algorithms for sparse matrices: Multiplication and permuted transposition, ACM Trans. Math. Softw. 4 (1978) 250–269.

[62] M. Deveci, C. Trott, S. Rajamanickam, Multi-threaded sparse matrix-matrix multiplication for many-core and gpu architectures, Parallel Computing 78 (2018).

[63] A. Buluc, J. Gilbert, Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments, SIAM Journal on Scientific Computing 34 (2011).

[64] Y. Saad, M. H. Schultz, GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems, SIAM Journal on Scientific and Statistical Computing 7 (1986).

[65] W. Yang, K. Li, Z. Mo, K. Li, Performance optimization using partitioned spmv on gpus and multicore cpus, IEEE Transactions on Computers 64 (2015) 2623–2636.

[66] T. Vejchodský, P. Šolín, Static condensation, partial orthogonalization of basis functions, and ILU preconditioning in the hp-FEM, Journal of Computational and Applied Mathematics 218 (2008) 192–200. doi:10.1016/j.cam.2007.04.044.

[67] D. A. Kopriva, E. Jimenez, An assessment of the efficiency of nodal discontinuous Galerkin spectral element methods, in: Recent Developments in the Numerics of Nonlinear Hyperbolic Conservation Laws, Springer, 2013, pp. 223–235.

[68] I. Huismann, L. Haupt, J. Stiller, J. Fröhlich, Sum factorization of the static condensed Helmholtz equation in a three-dimensional spectral element discretization, PAMM 14 (2014). doi:10.1002/pamm.201410465.

[69] D. A. Kopriva, Implementing spectral methods for partial differential equations: Algorithms for scientists and engineers, Springer Science & Business Media, 2009.