Two-level dynamic load-balanced p-adaptive discontinuous Galerkin methods for compressible CFD simulations

Yongseok Jang, Emeric Martin, Jean-Baptiste Chapelier, Vincent Couaillier

▶ **To cite this version:**

Yongseok Jang, Emeric Martin, Jean-Baptiste Chapelier, Vincent Couaillier. Two-level dynamic load-balanced p-adaptive discontinuous Galerkin methods for compressible CFD simulations. Computers & Mathematics with Applications, 2024, 176, pp.165-178. 10.1016/j.camwa.2024.10.008 . hal-04589705v2

HAL Id: hal-04589705
https://hal.science/hal-04589705v2

Submitted on 17 Feb 2025

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives 4.0 International License

Computers and Mathematics with Applications 176 (2024) 165–178

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

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

Two-level dynamic load-balanced p-adaptive discontinuous Galerkin methods for compressible CFD simulations

Yongseok Jang *, Emeric Martin, Jean-Baptiste Chapelier, Vincent Couaillier

DAAA, ONERA–The French Aerospace Lab, Polytechnic Institute of Paris, Châtillon, 92320, France

ARTICLE INFO

Keywords:
Dynamic load balancing
p-adaptation
High-order discontinuous Galerkin method
Two-level parallelization
CFD

ABSTRACT

We present a novel approach utilizing two-level dynamic load balancing for *p*-adaptive discontinuous Galerkin (DG) methods in compressible Computational Fluid Dynamics (CFD) simulations. The high-order explicit first stage, specifically the singly diagonal implicit Runge–Kutta (ESDIRK) method, is employed for time integration, where the pseudo-transient continuation is integrated with the restarted generalized minimal residual (GMRES) method to handle the solution of nonlinear equations at each stage of ESDIRK, excluding the initial stage. Relying on smoothness indicators, we carry out the refinement/coarsening process for *p*-adaptation with dynamic load balancing. This approach involves a coarse level (distributed memory) decomposition based on MPI paradigm and a fine level (shared memory) decomposition based on OpenMP paradigm, enhancing parallel efficiency. Dynamic load balancing is achieved by computing weights based on degrees of freedom, ensuring balanced computational loads across processors. The parallel computing framework adopts either a graph-based type (ParMETIS and Zoltan) or space-filling curves type (GeMPa) for coarse level partitioning, and a graph-based type (METIS and Zoltan) for fine level partitioning. The effectiveness of the method is demonstrated through numerical examples, highlighting its potential to significantly improve the scalability and efficiency of compressible flow simulations. The numerical simulations were conducted using the CODA flow solver, a state-of-the-art tool developed collaboratively by the French National Aerospace Center (ONERA), the German Aerospace Center (DLR), and Airbus.

1. Introduction

Computational Fluid Dynamics (CFD) has emerged as a powerful tool for simulating and analyzing fluid flow phenomena in various engineering and scientific applications. The ability to accurately predict fluid behavior plays a crucial role in optimizing designs, improving performance, and understanding complex flow phenomena. With advances in numerical methods and computational resources, CFD has become an indispensable tool in diverse fields such as aerospace, automotive, energy, and environmental engineering.

Developing efficient and accurate numerical methods capable of handling a wide range of flow conditions, ranging from laminar to turbulent and from subsonic to hypersonic regimes, remains a critical challenge in CFD. Significant research conducted in the late 1990s and early 2000s established Discontinuous Galerkin (DG) methods [1–3] as a powerful approach for addressing complex flow physics, including shock waves, boundary layers, and vortical structures, with high accuracy and efficiency. Similarly to finite volume schemes, DG discretizations can be solved in conservative form, which preserves conservation for mass, momentum and total energy. Furthermore, DG methods provide several advantages, such as high-order accuracy, geometric flexibility, and robustness on unstructured grids, making them a preferred choice for modern CFD applications.

Compared to continuous Galerkin finite element method and finite volume method, one of the primary weaknesses of DG methods lies in their computational cost, especially for high-order approximations and fine-grid resolutions. The discontinuous nature of the solution representation introduces additional numerical fluxes and inter-element communication, leading to increased computational overhead. To resolve the high computational cost and large memory requirement, Babůshka and Suri [4] introduced the adaptive method to adjust locally the spatial mesh h and/or the polynomial degree p. We also refer to the works of Houston and Süli [5,6] and the work of Hartmann and Houston [7]. While both h- and p-adaptations lead to reduction of the total degrees of

* Corresponding author.
E-mail addresses: yongseok.jang@onera.fr (Y. Jang), emeric.martin@onera.fr (E. Martin), jean-baptiste.chapelier@onera.fr (J.-B. Chapelier), vincent.couaillier@onera.fr (V. Couaillier).

https://doi.org/10.1016/j.camwa.2024.10.008
Received 5 June 2024; Received in revised form 20 September 2024; Accepted 6 October 2024
Available online 14 October 2024
0898-1221/© 2024 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

freedom during simulations, p-adaptation is often preferred in practice due to its ability to achieve high accuracy with lower computational cost, flexibility in targeting specific regions, preservation of conservation properties, and simplicity of implementation. Furthermore, the selection of p-adaptation or hp-adaptation depends on a careful consideration of the trade-offs between computational cost, implementation complexity, and the requirements of the problem at hand. Although hp-adaptation offers the potential for greater flexibility and accuracy by allowing both mesh refinement and polynomial degree adjustment, p-adaptation remains a viable and efficient approach in many practical applications.

In essence, error indicators are important components of adaptive algorithms, providing critical information for refining numerical solutions, optimizing computational resources, and ensuring accurate and efficient simulations across various domains and problem types. Three types of indicators are commonly used in the literature: (i) feature based [8,9], (ii) discretization error based [7,10], and (iii) adjoint-based [11,12] indicators. A comparative study by Naddei et al. [9] explored feature-based and discretization error-based indicators, concluding that *smoothness* indicators [13,8,14] demonstrate favorable numerical performance in terms of accuracy and efficiency. Conversely, adjoint-based methods involve significant computational costs, despite exhibiting superiority in steady problems, as observed in [15].

Load balancing is another critical aspect of parallel CFD simulations, especially for large-scale computations on distributed memory architectures, as it helps prevent idle processes and minimize communication overhead. Load balancing techniques aim to distribute computational work evenly across processors to maximize parallel efficiency and minimize execution time. In the context of p-adaptive DG methods, dynamic load balancing becomes even more challenging due to the varying computational workload associated with adaptive refinement. Recently, Li et al. [16] presented the dynamically load-balanced p-adaptive DG method for Euler equations on tetrahedral grids. Furthermore, the work of Jägersküpper and Vollmer [17] exhibited high parallel scalability of CFD on unstructured meshes based on a two-level domain decomposition.

Throughout our article, we primarily utilize the CFD software jointly developed by ONERA, DLR and Airbus, hereafter referred to as CODA [18,19], for numerical simulations. The CODA solver represents the new generation of CFD solvers for aeronautical applications. It includes both classic finite volume methods and modern high-order discontinuous Galerkin schemes, offering a versatile tool specifically tailored for the complex demands of aerodynamic design and research. CODA supports multidisciplinary analysis and design optimization, making it a multifunctional tool for research activities and aircraft design. As the new reference solver for aerodynamic applications within Airbus, including aircraft, helicopters, space, and military domains, CODA is set to play a pivotal role in enhancing the design and analysis processes across these diverse engineering fields. The solver's comprehensive capabilities extend from steady-state Reynolds-averaged Navier-Stokes (RANS) equations with various turbulence models to Large Eddy Simulation (LES) and hybrid RANS/LES models, catering to both steady and unsteady flow scenarios with a selection of explicit and implicit time-stepping methods. This makes CODA a powerful and flexible tool for tackling the most challenging aerodynamic simulations.

Our present work focuses on the development and application of two-level dynamic load-balanced p-adaptive DG methods for simulating unsteady compressible flows. The p-adaptive methods, which dynamically adjust the polynomial order of the solution within each element based on the local error estimate, offer a promising approach to achieve high accuracy while minimizing computational cost. By adaptively refining the local polynomial degrees in regions of high gradients or flow features and coarsening others, p-adaptive methods can capture fine-scale flow structures and improve solution accuracy without increasing computational overhead. Furthermore, to balance computational loads during simulations, several types of parallel mesh partitioners are introduced such as *graph*-[20], *hyper graph*-[21] and *space-filling curves* (SFC) [22] based partitioning. To the best of our knowledge, our research on two-level partitioning, combining the graph-type and the SFC-type mesh partitioners, is the first investigation to exhibit higher scalability.

The main objectives of this study are to develop robust and efficient p-adaptive algorithms and to investigate their performance and scalability on parallel computing platforms. The proposed methods will be validated against analytical solutions and benchmark test cases to assess their accuracy and robustness. Additionally, numerical experiments will be conducted to analyze the impact of dynamic load balancing strategies on parallel efficiency and scalability for a range of flow scenarios and computational configurations.

The article is organized as follows. Section 2 introduces compressible flow models with DG discretization as well as temporal discrete schemes. The p-adaptive algorithm with two-level dynamic load balancing is presented in Section 3. Various numerical results validate the accuracy and efficiency of our proposed method with the comparison study of load balancers in Section 4. Finally, Section 5 concludes our work.

2. Model problem

In this section, we present semi-discrete and fully discrete formulations for unsteady compressible flows. We consider a compressible Navier-Stokes equation on the bounded domain $\Omega \in \mathbb{R}^d$ where d is the space dimension. When we denote $\boldsymbol{u}$ as the conservative state variables such that $\boldsymbol{u} = (\rho, \rho \boldsymbol{V}^T, \rho E)^T$ with the density ρ, the velocity vector $\boldsymbol{V} = (V_1, V_2, V_3)^T$ and the total energy E, the compressible Navier-Stokes equation can be written as

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \underline{\boldsymbol{F}}^c(\boldsymbol{u}) - \nabla \cdot \underline{\boldsymbol{F}}^d(\boldsymbol{u}, \nabla \boldsymbol{u}) = \boldsymbol{0}, \tag{2.1}$$

where $\underline{\boldsymbol{F}}^c$ and $\underline{\boldsymbol{F}}^d$ represent convective and diffusive fluxes, respectively. The specific total energy E is defined as $E = P/\left(\rho(r_T - 1)\right) + \boldsymbol{V} \cdot \boldsymbol{V}/2$ with the pressure P and the specific heat ratio r_T. With Einstein notation, the convective and diffusive fluxes can be expressed by

$$\boldsymbol{F}_i^c(\boldsymbol{u}) = \begin{bmatrix} \rho V_i \\ \rho V_1 V_i + P\delta_{i1} \\ \rho V_2 V_i + P\delta_{i2} \\ \rho V_3 V_i + P\delta_{i3} \\ \rho V_i E \end{bmatrix} \quad \text{and} \quad \boldsymbol{F}_i^d(\boldsymbol{u}, \nabla \boldsymbol{u}) = \begin{bmatrix} 0 \\ \tau_{i1} \\ \tau_{i2} \\ \tau_{i3} \\ \tau_{ik} V_k - q_i \end{bmatrix}, \tag{2.2}$$

where the viscous stress tensor $\underline{\boldsymbol{\tau}}$ and the heat flux $\boldsymbol{q}$ are given by

$$\tau_{ij} = \mu \left(\frac{\partial V_i}{\partial x_j} + \frac{\partial V_j}{\partial x_i} - \frac{2}{3} \frac{\partial V_k}{\partial x_k} \delta_{ij} \right) \quad \text{and} \quad q_i = -\lambda \frac{\partial T}{\partial x_i}.$$

Here, μ is the dynamic viscosity, λ is the thermal conductivity defined with the specific heat capacity at constant pressure C_P and the Prandtl number Pr such as $\lambda = \mu C_P/\mathrm{Pr}$, and T denotes the temperature. We suppose that μ and λ are constants.

2.1. Discontinuous Galerkin method

Let the spatial domain Ω be subdivided by an unstructured mesh $\mathcal{E}_h$ consisting of non-overlapping and non-empty elements K. We denote the set of interior faces in $\mathcal{E}_h$ and the boundary faces by Γ_h and $\Gamma_b := \cup_{K \in \mathcal{E}_h} \partial K \backslash \Gamma_h$, respectively. When we define the space of polynomials of degree less than or equal to p over K such that

$$\mathcal{P}_p(K) = \mathrm{span} \left\{ x_1^{i_1} \cdots x_d^{i_d} \,|\, \sum_{m=1}^{d} i_m \le p,\ \boldsymbol{x} \in K, i_m \in \mathbb{N} \cup \{0\} \text{ for each } m \right\},$$

our DG finite element space is given by

$$D_k(\mathcal{E}_h) = \left\{ v \in L_2(\mathcal{E}_h) \,\middle|\, v|_K \in \mathcal{P}_p(K) \text{ for each } K \in \mathcal{E}_h \right\},$$

and the analogous vector field is defined by $\boldsymbol{\mathcal{V}}_k^h := [D_k(\mathcal{E}_h)]^{(d+2)}$.

By introducing appropriate variational projections for the convective and diffusive terms, we can derive the following semi-discrete problem: find $\boldsymbol{u}_h(t) \in \mathcal{V}_k^h$ satisfying for any $\boldsymbol{v} \in \mathcal{V}_k^h$,

$$\sum_{K\in\mathcal{E}_h} \int_K \frac{\partial \boldsymbol{u}_h}{\partial t} \cdot \boldsymbol{v} dK + \mathcal{L}^c(\boldsymbol{u}_h, \boldsymbol{v}) + \mathcal{L}^d(\boldsymbol{u}_h, \boldsymbol{v}) = 0, \tag{2.3}$$

where $\mathcal{L}^c$ and $\mathcal{L}^d$ are the variational forms of the convection and diffusion terms, respectively. The semi-discrete solution $\boldsymbol{u}_h$ can be written as

$$\boldsymbol{u}_h(t) = \sum_{i=1}^{N_{\text{dof}}} \boldsymbol{\phi}_i U_i(t), \tag{2.4}$$

where $\{\boldsymbol{\phi}_i\}$ is a set of basis functions of $\mathcal{V}_k^h$, N_{dof} is the number of degrees of freedom and $U_1, \dots, U_{N_{\text{dof}}}$ are degrees of freedom of the discrete solution. We refer to [3] for the procedure of generating orthonormal basis functions.

For an interior face $e = \partial K_i \cap \partial K_j$ with $i < j$, we define an average and a jump by

$$\{\boldsymbol{v}\} = \frac{(\boldsymbol{v}|_{K_i})|_e + (\boldsymbol{v}|_{K_j})|_e}{2} \qquad \text{and} \qquad [\boldsymbol{v}] = (\boldsymbol{v}|_{K_i})|_e - (\boldsymbol{v}|_{K_j})|_e.$$

For convenient notation, we denote $(\boldsymbol{v}|_{K_i})|_e$ and $(\boldsymbol{v}|_{K_j})|_e$ by '$\boldsymbol{v}^+$' and '$\boldsymbol{v}^-$', respectively. Using the jump and average, we introduce the variational formulations of the convective and viscous terms.

Upwind scheme In this paper, to define $\mathcal{L}^c$, we use a numerical flux based on *Roe scheme* [23,24]. By integrating by parts, we get

$$\begin{aligned}\mathcal{L}^c(\boldsymbol{u}_h, \boldsymbol{v}) = &- \sum_{K\in\mathcal{E}_h} \int_K \underline{\boldsymbol{F}}^c : \nabla \boldsymbol{v} dK + \int_{\Gamma_h} [\boldsymbol{v}] \cdot H_c(\boldsymbol{u}^+, \boldsymbol{u}^-, \boldsymbol{n}) de \\ &+ \int_{\Gamma_b} \boldsymbol{v} \cdot \underline{\boldsymbol{F}}^c(\boldsymbol{u}_b) \cdot \boldsymbol{n} de,\end{aligned} \tag{2.5}$$

where the notation: is for the tensor inner product, H_c is a numerical flux which approximates the convective flux on an element face, $\boldsymbol{n}$ is an outward normal vector and $\boldsymbol{u}_b$ is given by the boundary condition. We refer to [23] for the detail of the numerical flux H_c of the Roe scheme.

Lifting operator By introducing lifting operators, we can discretize the diffusion term. In particular, we employ local and global lifting operators proposed by Bassi and Rebay, so-called *BR2* method, e.g., see [1,2,25]. We define a local lifting operator $\underline{\boldsymbol{l}}_h^e$ for $e \in \Gamma_h$ that satisfies

$$\int_{K_i \cup K_j} \underline{\boldsymbol{l}}_h^e \cdot \boldsymbol{v} dK = -\int_e [\boldsymbol{u}_h \otimes \boldsymbol{n}] \cdot \{\boldsymbol{v}\} de, \qquad \forall \boldsymbol{v} \in \mathcal{V}_k^h, \tag{2.6}$$

where $e = \partial K_i \cap \partial K_j$ of the distinct two elements K_i and K_j. Here, the support of the local lifting operator is the union of neighbor elements of e. Then a global lifting operator $\underline{\boldsymbol{L}}_h^K$ is given as the sum of local lifting operators such that

$$\underline{\boldsymbol{L}}_h^K = \sum_{e\in\partial K} \underline{\boldsymbol{l}}_h^e \qquad \text{for each } K. \tag{2.7}$$

Using the definition of the lifting operators, we can derive the variational form of the diffusive term such that

$$\begin{aligned}\mathcal{L}^d(\boldsymbol{u}_h, \boldsymbol{v}) = &\sum_{K\in\mathcal{E}_h} \int_K \underline{\boldsymbol{F}}^d(\boldsymbol{u}, \nabla \boldsymbol{u} + \underline{\boldsymbol{L}}_h^K) : \nabla \boldsymbol{v} dK \\ &- \int_{\Gamma_h} [\boldsymbol{v}] \cdot \{\underline{\boldsymbol{F}}^d(\boldsymbol{u}, \nabla \boldsymbol{u} + \eta_e \underline{\boldsymbol{l}}_h^e)\} \cdot \boldsymbol{n} de \\ &- \int_{\Gamma_b} \boldsymbol{v} \cdot \underline{\boldsymbol{F}}^d(\boldsymbol{u}_b, \nabla \boldsymbol{u}_b + \eta_e \underline{\boldsymbol{l}}_h^e) \cdot \boldsymbol{n} de,\end{aligned} \tag{2.8}$$

where η_e is the number of faces of neighbor elements.

Let $\boldsymbol{U}(t)$ be a vector of degrees of freedom. Then the semi-discrete problem (2.3) becomes equivalent to the following ODE system:

$$M \frac{d\boldsymbol{U}}{dt} + \boldsymbol{\mathcal{R}}(\boldsymbol{U}) = \boldsymbol{0}, \tag{2.9}$$

where M is the mass matrix and $\mathcal{R}$ is the spatial residual vector. Assembling M and $\boldsymbol{\mathcal{R}}(\boldsymbol{U})$ follows an obvious way such that $\forall i, j = 1, \dots, N_{\text{dof}}$,

$$M_{ij} = \sum_{K\in\mathcal{E}_h} \int_K \boldsymbol{\phi}_j \cdot \boldsymbol{\phi}_i dK \qquad \text{and} \qquad \mathcal{R}_i(\boldsymbol{U}) = \mathcal{L}^c(\boldsymbol{u}_h, \boldsymbol{\phi}_i) + \mathcal{L}^d(\boldsymbol{u}_h, \boldsymbol{\phi}_i).$$

Note that the system of ODEs (2.9) is nonlinear hence further linearization would be required to solve it numerically.

2.2. Time integration

Let $\Delta t > 0$ be a (global) time step size. We introduce inner-outer time integration schemes to obtain a fully discrete problem of (2.9). The inner time integration is employed for linearizing the spatial residual vector, while the outer time integration is performed to compute a fully discrete solution for the next time step. We mainly concern the *linearly implicit Euler method* for linearization. As our outer time integrator, we consider implicit Runge-Kutta-type time integrations. In particular, we employ *fourth order explicit singly diagonally implicit Runge-Kutta* (ESDIRK4) scheme [26]. We also refer to [27] for more details of the implicit Runge-Kutta methods.

Implicit Euler method We employ the implicit Euler scheme in (2.9) for linearization to yield

$$\left(\frac{1}{\Delta t} M + J^n\right) \Delta \boldsymbol{U}^n = -\boldsymbol{\mathcal{R}}(\boldsymbol{U}^n), \tag{2.10}$$

where $\Delta \boldsymbol{U}^n = \boldsymbol{U}^{n+1} - \boldsymbol{U}^n$ and J^n is the Jacobian matrix defined by

$$J^n = \frac{\partial \boldsymbol{\mathcal{R}}(\boldsymbol{U}^n)}{\partial \boldsymbol{U}}.$$

In addition, we use the pseudo-transient continuation technique, namely *Switched Evolution Relaxation* (SER) algorithm [28] and the *Jacobian-free Newton Krylov* (JFNK) method [29].

The SER algorithm leads to quadratic convergence of nonlinear iterations by introducing pseudo-time stepping and it brings

$$\left(\frac{1}{\Delta \tau^k} M + J^{n,k}\right) \Delta \boldsymbol{U}^k = -\boldsymbol{\mathcal{R}}(\boldsymbol{U}^{n,k}), \tag{2.11}$$

where $\Delta \boldsymbol{U}^k := \boldsymbol{U}^{n,k+1} - \boldsymbol{U}^{n,k}$, $\boldsymbol{U}^{n,0} = \boldsymbol{U}^n$, $\Delta \tau^k$ is the pseudo-time step size defined by

$$\Delta \tau^0 = \text{CFL}_{\min} \qquad \text{and} \qquad \Delta \tau^{k+1} = \min\left(\Delta \tau^k \frac{\left\|\boldsymbol{\mathcal{R}}(\boldsymbol{U}^{n,k-1})\right\|_{L_2}^{\beta}}{\left\|\boldsymbol{\mathcal{R}}(\boldsymbol{U}^{n,k})\right\|_{L_2}^{\beta}}, \text{CFL}_{\max}\right),$$

for user-defined parameters $0 < \text{CFL}_{\min} < \text{CFL}_{\max}$, and $0 < \beta \le 1$. Here $\Delta \tau^k$ is often referred to as the CFL number at k-th nonlinear step and is also called a local time step size. Note that too large CFL numbers may cause the linear systems to be ill-conditioned.

In solving the linear algebraic system corresponding to (2.11), the computation of $J^{n,k}$ is expensive. The computation of the Jacobian appears only once in the matrix-vector product if we use Krylov subspace methods to solve the linear system. Hence, instead of matrix-vector multiplication for the exact Jacobian matrix, we will employ the Jacobian-free approach of the automatic differentiation type [30]. This Jacobian matrix-free method is beneficial in computational costs as well as memory reductions. Consequently, we can solve the linear system by using the preconditioned GMRES method. In this study, we consider various linear solvers with combinations such as block Jacobi preconditioner, block ILU(0) preconditioner, and block element LU decomposition.

Table 2.2.1
The standard Butcher tableau of 4 stages (left) and the Butcher tableau of ESDIRK43 (right) [27].

c_1	a_{11}	0	0	0
c_2	a_{21}	a_{22}	0	0
c_3	a_{31}	a_{32}	a_{33}	0
c_4	a_{41}	a_{42}	a_{43}	a_{44}
	b_1	b_2	b_3	b_4

0	0	0	0	0
2γ	γ	γ	0	0
1	$1-a_{32}-\gamma$	a_{32}	γ	0
1	$1-b_2-b_3-\gamma$	b_2	b_3	γ
	$1-b_2-b_3-\gamma$	b_2	b_3	γ

ESDIRK scheme We consider the ESDIRK method (in particular ESDIRK43) of four stages (consisting of a first explicit stage and three implicit stages) and a fourth order. The ESDIRK43 scheme follows:

$$\begin{cases} \boldsymbol{M}\boldsymbol{U}^{n+1} = \boldsymbol{M}\boldsymbol{U}^{n} - \Delta t \sum_{i=1}^{4} b_i \boldsymbol{\mathcal{R}}(\boldsymbol{Y}^{(i)}), \\ \boldsymbol{M}\boldsymbol{Y}^{(i)} = \boldsymbol{M}\boldsymbol{U}^{n} - \Delta t \sum_{j=1}^{i} a_{ij} \boldsymbol{\mathcal{R}}(\boldsymbol{Y}^{(j)}) \qquad \text{for } i = 1, \dots, 4, \end{cases} \tag{2.12}$$

where the set of coefficients is formed by the Butcher tableau illustrated in Table 2.2.1. To complete the fill of Runge-Kutta coefficients, we set

$$\gamma = \text{one of the approximate roots of } 6x^3 - 18x^2 + 9x - 1 = 0,$$

$$a_{32} = \frac{-2\gamma + 1}{4\gamma}, \qquad b_2 = \frac{-1}{12\gamma(2\gamma - 1)}, \qquad b_3 = \frac{-6\gamma^2 + 6\gamma - 1}{6\gamma - 3}.$$

We refer to [31,27] for the definition of Butcher tableau for (ES)DIRK schemes and their properties.

In the ESDIRK scheme (2.12), it is necessary to solve three nonlinear systems for $\boldsymbol{Y}^{(2)}$, $\boldsymbol{Y}^{(3)}$ and $\boldsymbol{Y}^{(4)}$, since $a_{11} = 0$. These three nonlinear systems can be solved by recalling the implicit Euler method with the pseudo-transient continuation (2.11). To be specific, the nonlinear system in (2.12) can be rewritten as

$$\boldsymbol{\mathcal{F}}^n(\boldsymbol{Y}^{(i)}) := \frac{1}{\Delta t}\boldsymbol{M}\boldsymbol{Y}^{(i)} + \gamma \boldsymbol{\mathcal{R}}(\boldsymbol{Y}^{(i)}) - \frac{1}{\Delta t}\left(\boldsymbol{M}\boldsymbol{U}^n - \Delta t \sum_{j=1}^{i-1} a_{ij} \boldsymbol{\mathcal{R}}(\boldsymbol{Y}^{(j)}) \right) = 0, \tag{2.13}$$

for $i = 2, 3, 4$, since $a_{ii} = \gamma$. Let $\boldsymbol{Y}^{i,m}$ be an approximate solution of $\boldsymbol{Y}^{(i)}$ at m-th iteration and $\Delta \boldsymbol{Y}^{i,m} := \boldsymbol{Y}^{i,m+1} - \boldsymbol{Y}^{i,m}$. After noting that the Jacobian matrix of $\boldsymbol{\mathcal{F}}^n(\boldsymbol{Y}^{i,m})$ is defined as

$$\frac{\partial \boldsymbol{\mathcal{F}}^n(\boldsymbol{Y}^{i,m})}{\partial \boldsymbol{Y}} = \frac{1}{\Delta t}\boldsymbol{M} + \gamma \boldsymbol{J}^{i,m} \qquad \text{where } \boldsymbol{J}^{i,m} = \frac{\partial \boldsymbol{\mathcal{R}}(\boldsymbol{Y}^{i,m})}{\partial \boldsymbol{Y}},$$

the pseudo-transient continuation on (2.13) leads to

$$\left(\frac{1}{\Delta t}\boldsymbol{M} + \frac{1}{\Delta \tau^m}\boldsymbol{M} + \gamma \boldsymbol{J}^{i,m} \right) \Delta \boldsymbol{Y}^{i,m} = -\boldsymbol{\mathcal{F}}^n(\boldsymbol{Y}^{i,m}). \tag{2.14}$$

As seen in the Euler scheme, (2.14) will be solved by the JFNK method for $Y^{(2)}$, $Y^{(3)}$ and $Y^{(4)}$ one at a time.

3. Dynamic load-balanced p-adaptation

Achieving high-performance simulations involves dynamically balanced computational loads across the domains, while jointly adjusting the polynomial degree p for precise and resource-efficient solutions. In this section, we explore dynamic load-balanced p-adaptation, a fundamental component of our p-adaptive DG algorithm, to enhance the scalability and parallel efficiency of the simulation. We investigate the collaborative integration of load balancing and p-adaptation, presenting a comprehensive overview of the algorithm, its adaptive strategy utilizing '*smoothness*' based indicators, and the effective ways of repartitioning computational domains. Furthermore, hybrid parallel computation, blending distributed memory and shared memory improves scalability as well as efficiency. Positioned at the forefront of advanced CFD techniques, this adaptive methodology provides a scalable solution to meet the computational demands of modern simulations.

Algorithm 1 Two-level parallel p-adaptive DG algorithm in CFD simulation.

User defined parameters: the number of time steps N, global time step size Δt, pseudo-time stepping parameters ($\text{CFL}_{\min}$, $\text{CFL}_{\max}$, β), frequency of refinement N_p.
Setup a physical model problem.
Initialize spatial and temporal discretizations, boundary conditions, and mesh partitioning.
Compute the discrete solution at the initial state $n = 0$.
while $n \le N$ **do**
  Solve the discrete problem of (2.9) w.r.t. the time integrator.
  $n \leftarrow n + 1$
  if mod $(n, N_p) = 0$ and $n < N$ **then**
    Evaluate *a posteriori* indicator η_K for each element K.
    Refine and coarsen the polynomial degree p for each K
    Define weight values based on p for element.
    Partition the mesh w.r.t. weight values for distributed memory level.
    Partition subdomains for shared memory level w.r.t. weight values.
  end if
end while

We provide the p-adaptive DG algorithm with two-level dynamic load balancing for unsteady CFD simulation in Algorithm 1. Varying with time integrator, we solve different fully discrete problems such as (2.12). Due to the presence of pseudo-transient continuation, ESDIRK scheme consists of inner-outer time integration for non-linear steps and linear steps, respectively. Hence the ESDIRK scheme requires two stopping criteria for nonlinear iterations and linear iterations, when the Krylov subspace method is employed as the linear solver.

Depending on the choice of p-refinement strategies and mesh partitioning methods, our algorithm will have different numerical performance. In this study, we consider the p-adaptation based on smoothness indicators. To implement the dynamically load-balanced adaptive algorithm, we present and compare different types of parallel mesh partitioning.

3.1. A posteriori error indicator

The smoothness indicator is widely used for shock capturing [13]. This is beneficial in stabilizing the discontinuities during simulations. We define three smoothness refinement indicators; *small-scale energy density* (SSED) indicator [14], *spectral decay* (SD) indicator [8] and *Persson&Peraire* (PP) indicator [13]. These element-wise indicators are computed for each $K \in \mathcal{E}_h$ by

$$\eta_K^{\text{SSED}} = \frac{\left\| (\rho \boldsymbol{V})_p - (\rho \boldsymbol{V})_{p-1} \right\|_{L_2(K)}}{\sqrt{|K|}}, \tag{3.1}$$

$$\eta_K^{\text{SD}} = \frac{\left\| (\rho \boldsymbol{V})_p - (\rho \boldsymbol{V})_{p-1} \right\|_{L_2(K)}}{\left\| (\rho \boldsymbol{V})_p \right\|_{L_2(K)}}, \tag{3.2}$$

$$\eta_K^{\text{PP}} = \frac{\left\| (\rho)_p - (\rho)_{p-1} \right\|^2_{L_2(K)}}{\left\| (\rho)_p \right\|^2_{L_2(K)}}, \tag{3.3}$$

where $(\cdot)_p$ denotes the numerical solution of polynomial degree p, $(\cdot)_{p-1}$ represents the projection of the discrete solution onto the lower order space $\mathcal{P}_{p-1}(K)$ and $|K|$ is the geometrical measure of K. While the SSED and SD indicators are defined with the momentum density to capture discontinuity in kinetic energy, the PP indicator is developed to detect cells for shock capturing. The performance of the SSED indicator is constrained by its dependence on the higher modes of the solution. The SD indicator underperforms due to its tendency to incorrectly identify regions with low values in both high-energy content and total energy [32]. On the other hand, the PP indicator is restricted to measuring energy er-

rors. The optimal choice of error indicators for p-adaptation is still an open question and remains problematic.

Maximum marking adaptive strategy After evaluating error indicators for all elements, we decide whether to increase, decrease or keep polynomial degrees for each element. When we denote the maximum indicator over elements by $\eta_{\max}$, the adaptation criteria in our study are given as follows:

$$\begin{cases} \text{increase the polynomial order } p \text{ by one} & \text{if } \eta_K \geq \nu_{\max}\eta_{\max}, \\ \text{but no higher than the maximum} & \\ \text{allowed degree } p_{\max}, & \\ \text{decrease the polynomial order } p \text{ by one} & \text{if } \eta_K \leq \nu_{\min}\eta_{\max}, \\ \text{but no lower than the minimum} & \\ \text{allowed degree } p_{\min}, & \end{cases}$$

where $\nu_{\min}$ and $\nu_{\max}$ are user-defined parameters. Additionally, if p is already at its limit $p_{\max}$ (or $p_{\min}$), no further increase (or decrease) in the polynomial degree will be made. In this case, p will either be maintained at its current value or, if necessary, adjusted to the nearest allowed value within the constraints. This ensures that p remains within the predefined bounds at all times. Taking lower values of $\nu_{\max}$ leads to more accurate orders of convergence but more expensive computational costs. Also, larger $\nu_{\min}$ implies aggressive coarsening in p.

3.2. Mesh partitioning

Mesh partitioning is a fundamental aspect of parallel computing in numerical simulations, particularly within the domain of CFD. This process involves dividing a computational mesh into multiple subdomains to enable efficient parallel processing on distributed computing systems. Mesh partitioning plays a crucial role in enhancing computational performance, minimizing communication overhead, and optimizing resource utilization in parallel and high-performance computing environments.

Various strategies are employed for mesh partitioning, each tailored to specific simulation requirements and the architecture of the computing system, for example, *recursive bisection* (RCB) method, graph-based methods, and SFC method. RCB divides the mesh into two halves recursively, while graph-based methods model the mesh as a graph, where nodes represent mesh elements, and edges represent connections. SFC methods leverage curve-based approaches, such as the Hilbert or Morton curve, to achieve optimal partitioning. The RCB is the most simple, fast and inexpensive but it is inefficient in practice, especially, with complex geometries. The RCB partitioning is nevertheless employed to balance the initial mesh-data distribution obtained after loading the mesh, to avoid memory bottleneck in the graph-based partitioning.

The scalability limits of graph-based partitioning algorithms become apparent as memory consumption grows linearly with the graph size. One such alternative is the SFC partitioning method. SFC maps the 1D unit interval onto a higher-dimensional space in a way that neighboring points on the unit interval also become neighboring points in the target space. This results in a partitioning approach characterized by low memory usage, offering a potential solution to address the scalability challenges observed in traditional graph-based methods. For the comparison of the methods, we refer to [22].

In this work, we employ several mesh partitioners such as METIS/ParMETIS [20], Zoltan [21] and GeMPa [33,34] as described in Table 3.2.1. The RCB will not be used solely as a mesh partitioner but it will be employed with graph-based partitioners for initial distribution. We thereby consider three partitioners for the MPI process level and two partitioners for the thread level. For example, using the corresponding mesh partitioner, we can distribute the unstructured mesh of NACA12 airfoil on 4 MPI processes as illustrated in Fig. 3.2.1. The mesh consists of 28,590 elements of prisms and hexahedrons and each partitioner is supposed to equally allocate the number of elements per process. We can observe the characteristics of bisection methods in RCB such that the domain is geometrically divided into 4 parts. The graph based partitioners, ParMETIS and Zoltan, are combined with RCB as the initial distributor and Zoltan exhibit the property of hypergraph partitioning. To be specific, the hypergraph partitioning of Zoltan allows Process 3 to possess elements of upper and lower parts for the airfoil where they are disconnected. On the other hand, GeMPa shows the different partitioning results that Process 1 does not contain any element near the airfoil. In terms of computational complexity, the cost of Zoltan is the most expensive but the cost of partitioning will be negligible in the total numerical simulation.

Table 3.2.1
Mesh partitioners.

Mesh partitioner	Type	Usage
RCB	Geometry	Initial distribution
ParMETIS	Graph-based	Distributed memory level
METIS	Graph-based	Shared memory level
Zoltan	Graph-based	Distributed and shared memory levels
GeMPa	SFC	Distributed memory level

Remark. ParMETIS is an extension of METIS designed for parallel computing environments. It provides scalable parallel graph partitioning algorithms that are crucial for large-scale simulations and computations on distributed memory parallel machines. Zoltan, like METIS and ParMETIS, is designed for parallel computing and additionally provides hypergraph partitioning to connect more than two vertices, offering a more expressive means to capture complex connectivity patterns beyond standard graphs. For a detailed evaluation of graph-based partitioners, we refer to [35].

Partitioning weights In the context of dynamic load balancing for p-adaptive simulations, the weight of each element is computed to achieve a balanced distribution across MPI processes. The weight value for an element is determined by the number of degrees of freedom corresponding to a given polynomial degree. For example, in 3D, the weight w_K for each element $K \in \mathcal{E}_h$ is calculated as follows:

$$w_K = \frac{(p+1)(p+2)(p+3)}{6}.$$

Let I_i denote the mesh partition distributed on MPI process i, where $\cup_{i=1}^{n_{\text{proc}}} I_i = \mathcal{E}_h$, and $I_i \cap I_j = \emptyset$ for $i \neq j$, with n_{proc} as the number of processes in the simulation. The distribution ensures that the sum of w_K for each MPI process fulfills the condition for the imbalance factor such that

$$\text{Imbalance factor} := \frac{n_{\text{proc}} w_{\max}}{w_{\text{total}}} \leq 1.05.$$

Here, $w_{\max}$ is the maximum local load, and w_{total} is the total load defined by:

$$w_{\max} = \max_i \sum_{K \in I_i} w_K \qquad \text{and} \qquad w_{\text{total}} = \sum_{K \in \mathcal{E}_h} w_K.$$

Consequently, this condition ensures that the load imbalance across domains stays within a 5% threshold concerning weights. Similarly, load balancing at the thread level is performed based on weight values to maintain this criterion for load balance.

3.3. Two-level parallel computing

In our specific implementation of two-level decomposition, MPI is utilized at the coarse level for inter-process communication, and OpenMP is used at the fine level for intra-process parallelizm. At the distributed memory level, the classical domain decomposition requires the use of *halo* cells to make remote data locally available, leading to an increase in the total memory footprint with the number of processes (n_{proc}) and limited parallel scalability. To address this, the goal is to

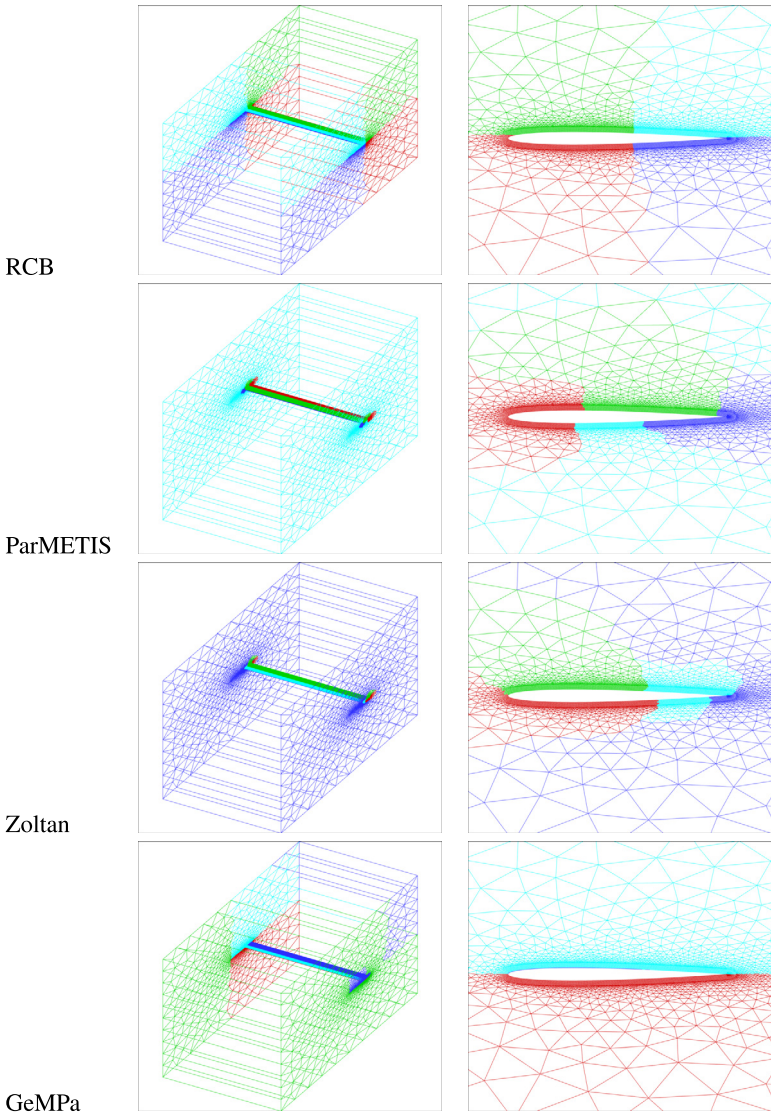


Fig. 3.2.1. Distribution of elements in processes 0-3 w.r.t. mesh partitioners: 3D global view (left) and XZ plane 2D close view around the NACA12 airfoil. Processes 0-3 are colored by red, green, blue and sky blue, respectively.

minimize synchronization and maximize data locality. On the shared memory level, multi-threading is employed for thorough subdomain decomposition, allowing each cell to be exclusively owned by a unique thread, thus reducing global MPI synchronization needs. Additional details on the hybrid parallelization of MPI and OpenMP can be found in [17].

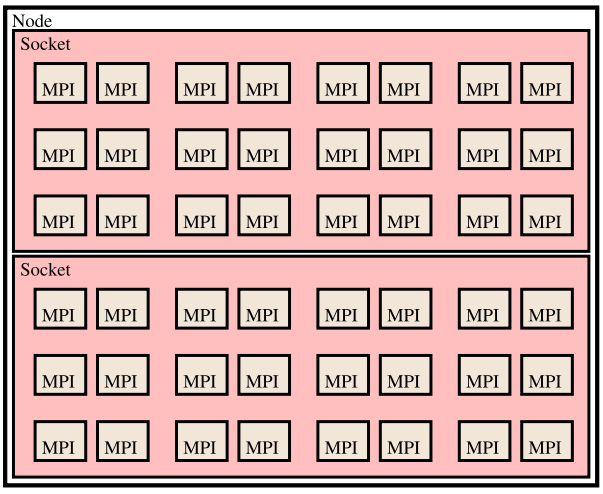
4. Numerical experiments

In this section, we present numerical examples to compare the proposed algorithm across various CFD simulations. The numerical simulations were conducted using the CODA and ONERA's cluster system, where each node is equipped with two Intel Xeon Cascade Lake - 6240R processors, featuring 24 cores (2.4 GHz, 35.75MB cache). To specify the (MPI) processes and threads configuration on a node, we use `P` and `T`, respectively. For instance, in MPI-only mode with one node having 48 physical cores, the configuration is denoted as 48`P` per node. In the case of the two-level partitioning with 12 MPI processes and 4 threads per node, it is expressed as 12`P` × 4`T`. The architecture of a node is depicted in Fig. 4.0.1. Depending on the number of cores used, we employ a corresponding number of nodes. For instance, when simulating numerical tests with 480 cores, either in MPI mode or hybrid mode, we use 10 nodes.

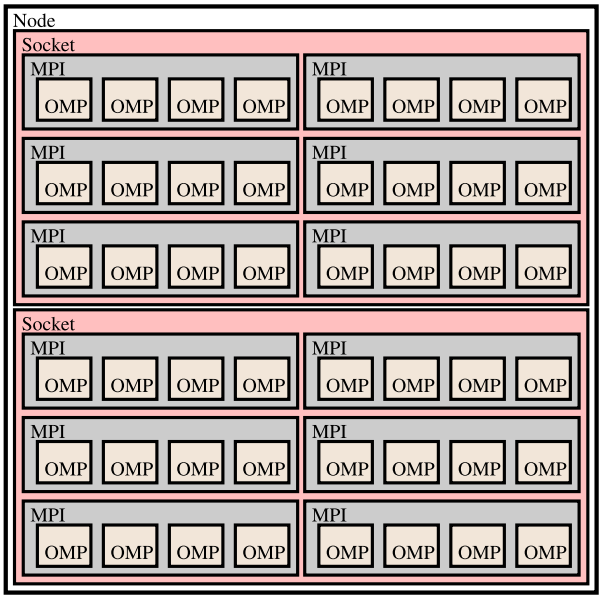
During the CFD simulation, we introduce two types of stopping criteria for nonlinear and linear iterations, respectively. For nonlinear steps, we define the maximum number of pseudo-Newton iterations N_{nl} per time step and the convergence tolerance of relative residual for variables of interest ϵ_{nl}, where the variables are defined as the density, the momentum and the energy. When we solve linear systems in our simulations, we employ GMRES-type solvers with the stopping criteria of the maximum number of linear iterations N_{l} and the relative linear residual error tolerance ϵ_{l}. We combine GMRES solvers with block Jacobi and LU-type preconditioners (element-wise LU and ILU0) for convergence. For simplicity, we denote linear solver setting as follows:

- GMRES(m): the restarted GMRES with the Krylov subspace size m.
- Jacobi(k): k-times applying the block Jacobi iteration.

Table 4.0.1 demonstrates the setting of p-adaptive solver parameters for each test case.



(a) Full MPI mode 48P



(b) Hybrid mode 12P × 4T

Fig. 4.0.1. Hardware topology of a `Cascade Lake` node with 24-cores sockets. Process and thread distribution where each brown box represents a physical core.

Table 4.0.1
Solver and parameter settings.

	Case 1	Case 2	Case 3
GMRES	GMRES(50)	GMRES(200)	GMRES(100)
Jacobi	Jacobi(10)	Jacobi(10)	Jacobi(10)
LU	ILU0	element-wise LU	element-wise LU
$\text{CFL}_{\min}$	10^6	1	1
$\text{CFL}_{\max}$	10^{12}	10^{12}	10^{12}
β	0.4	0.4	0.4
$(\epsilon_{\text{nl}}, N_{\text{nl}})$	$(10^{-8}, 200)$	$(10^{-5}, 200)$	$(10^{-4}, 200)$
$(\epsilon_{\text{l}}, N_{\text{l}})$	$(10^{-3}, 100)$	$(10^{-2}, 200)$	$(10^{-2}, 400)$
$(\nu_{\min}, \nu_{\max})$	$(0.001, 0.1)$	$(0.01, 0.5)$	$(0.01, 0.001)$
$(p_{\min}, p_{\max})$	$(1, 3)$	$(2, 4)$	$(1, 4)$
N_p	5	2	5

4.1. Unsteady Euler flow

As our first numerical example, the convection of a 2D isentropic vortex in a uniform flow is simulated on a 3D mesh (by taking one cell in z-direction). As described in [36], the vortex is transported by the uniform flow at a Mach number $\text{Ma} = 0.5$ in the x-direction. For the reference state of the Kok vortex, we define the reference density $\rho_\infty = 1$, the reference pressure $P_\infty = 1$, the specific heat ratio $r_T = 1.4$ and the radius of vortex $r = 0.5$. While we consider the computational domain $[-22.25, -14.25] \times [-4, 4] \times [0, 1]$, the initial center of the vortex is located at $(-18.75, 0, z)$ and the vortex travels 30% of domain during simulating. We define a uniform hexahedral mesh of $100 \times 100 \times 1$ elements and impose periodic boundary conditions on all boundaries. With the vortex transport period $8/(\text{Ma}\sqrt{\ln 2})$, we consider 2000 time steps and hence define the global time step size Δt by

$$\Delta t = 0.3 \times (\text{the vortex transport period})/(\text{the number of time steps}).$$

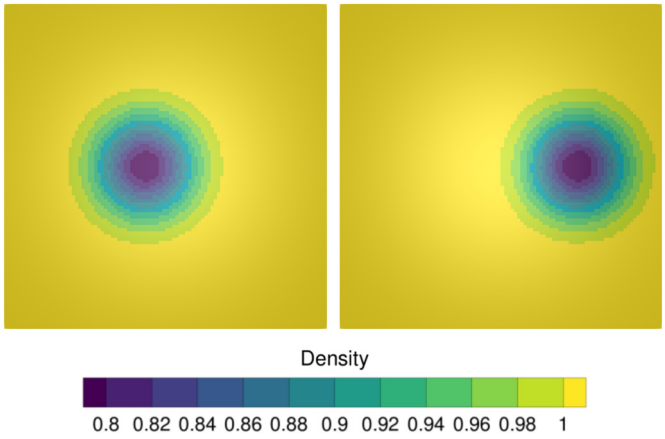


Fig. 4.1.1. Initial (left) and final (right) states of simulations with uniform DG(3).

Numerical results Following the illustration in Fig. 4.1.1, the vortex transportation is effectively captured by the uniform DG(3) scheme. Similarly, the p-adaptive DG solvers with ParMETIS in full MPI mode, as shown in Fig. 4.1.2, successfully preserve the vortex. Although the usage of PP indicator leads to less smoothness of the solution, the numerical density ensures the persistence of the vortex in the simulation. Fig. 4.1.3 illustrates the distribution of polynomial degrees with respect to indicators at the final state. Furthermore, as in Fig. 4.1.4, both SSED and SD indicators lead to similar p-refinements with more elements of higher polynomial orders than the PP indicator.

As shown in Table 4.1.1, the p-adaptive algorithm can significantly reduce the runtime of a simulation. In particular, the dynamically load-balanced (based on the weight values) p-adaptive solvers exhibit more efficiency than the static p-adaptive solvers where the subdivisions are distributed on processes to have the (almost) equal number of elements. Therefore, the load balancing process is essential in the p-adaptive algorithm for the sake of efficiency. While the different indicators lead to different p-refinements as well as numerical convergences, dynamic load balancing does not change the number of nonlinear/linear iterations. Utilizing a coarse-level distribution of 480 cores with ParMETIS, the SSED and SD indicators with load balancing result in an almost half reduction in run time compared to the DG(3) scheme, as detailed in Table 4.1.1. Notably, since the most refined elements have linear polynomial order, the ADG solver with the PP indicator demonstrates significant time reduction, e.g., more than 80%. Hereafter, we always combine ADG solvers with load balancing.

When we evaluate L_2 norm error of cell averaged quantities such as numerical density, momentum V_1 and V_2, and energy density ρE in Table 4.1.2, we can observe similar L_2 norm error results between the DG and ADG solvers. Although the PP indicator results in only $P1$ and $P2$ elements, the numerical solution exhibits reasonable numerical errors in cell based average computation.

Next, we examine the numerical scalability of the simulation on 1 million elements using up to 1960 cores. The numerical scalability rate can be computed by the change in runtime over the change in the num-

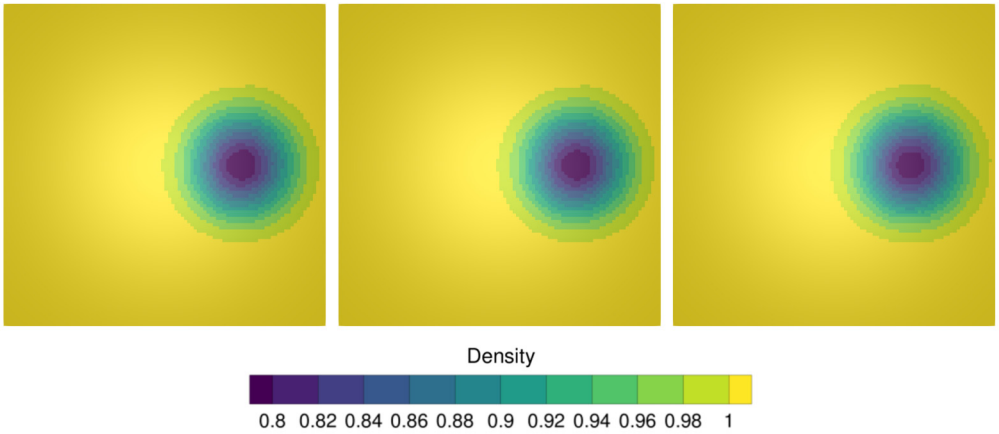


Fig. 4.1.2. Numerical density by p-adaptive solvers with SSED (left), SD (center) and PP (right) indicators for the final time.

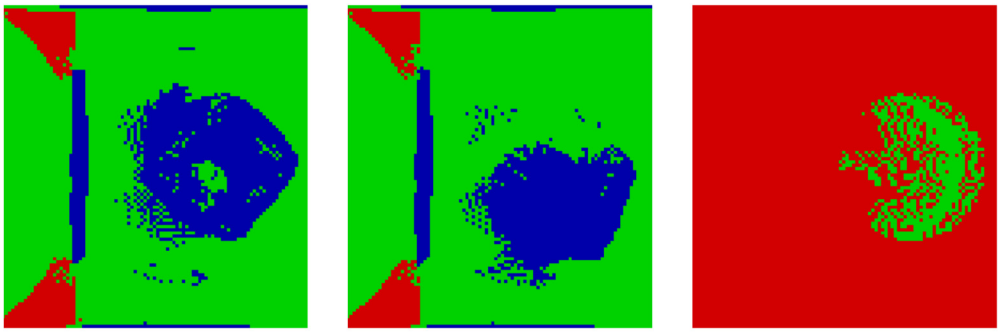


Fig. 4.1.3. Distribution of polynomial orders (e.g., red for $P1$, green for $P2$, and blue for $P3$). The figure shows p-adaptation with SSED (left), SD (center), and PP (right) indicators.

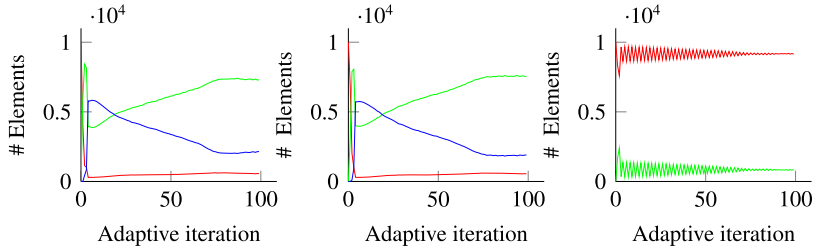


Fig. 4.1.4. Change of polynomial orders in adaptation with SSED (left), SD (center), and PP (right) indicators, which are colored by red for $P1$, green for $P2$, and blue for $P3$, respectively (x-axis: adaptive iterations and y-axis: the number of elements).

Table 4.1.1
CPU run time of simulations with 480 cores (if load-balanced, it is partitioned by ParMETIS in MPI mode).

	Total nonlinear It.	Total linear It.	Run time (s)	Reduction (%)
DG(3)	6142	8901	1884.08	
ADG-SSED	4515	6976	1755.79	6.81
(load-balanced)	4515	6976	1133.80	39.82
ADG-SD	3952	6417	1530.99	18.74
(load-balanced)	3952	6417	994.07	47.24
ADG-PP	3737	5308	310.42	83.52
(load-balanced)	3738	5307	153.31	91.86

Table 4.1.2
L_2 norm error of DG(3) and ADG schemes at the final time.

	Density ρ	Momentum V_1	Momentum V_2	Energy density ρE
DG(3)	7.931e-04	5.103e-04	3.207e-02	2.208e-03
ADG-SSED	7.925e-04	5.116e-04	3.206e-02	2.210e-03
ADG-SD	7.925e-04	5.115e-04	3.206e-02	2.210e-03
ADG-PP	7.908e-04	4.926e-04	3.195e-02	2.205e-03

Table 4.1.3
Scalability rates for ADG-SD solver simulations on 1 million mesh elements. MPI mode: 48P per node, hybrid: 12P × 4T per node with up to 1920 cores.

Coarse level	Fine level	Scalability rate
ParMETIS	(MPI only)	0.945
	METIS	0.972
	Zoltan	0.968
Zoltan	(MPI only)	0.946
	METIS	0.972
	Zoltan	0.976
GeMPa	(MPI only)	0.948
	METIS	0.961
	Zoltan	0.976

ber of cores so that the ideal rate is 1. Table 4.1.3 describes the numerical scalability rates with respect to mesh partitioning, where the simulations are performed with the ADG-SD solver for 1 million elements. While the MPI mode utilizes 48P on each node, the hybrid approach employs 12P × 4T in their simulations.

As a result, as seen in Table 4.1.3, the two-level partitioning displays better scalability for all coarse-level distributors than the MPI-only mode. In particular, using Zoltan as the shared memory distributor shows the most efficiency, with Zoltan and GeMPa for the distributed memory level, while ParMETIS is better combined with the shared memory partitioner METIS.

4.2. Unsteady Taylor-Green Vortex flow

Next, we consider the 3D compressible Navier-Stokes equation with the simple initial flow called the *Taylor-Green Vortex* (TGV). The flow is simulated on a periodic spatial domain $[-\pi, \pi]^3$ with 64,000 hexahedral elements. We refer to https://cfd.ku.edu/hiocfd/case_c3.5.pdf for physical properties and flow conditions so that we define the Mach number $Ma = 0.1$, the initial velocity $V_0 = 1$, the initial density $\rho_0 = 1$, $r_T = 1.4$, the initial temperature $T_0 = 273.15$, the Prandtl number $Pr = 0.71$ the pressure $p_0 = 1/(r_T Ma)^2$ and the Reynolds number $Re = 1600$. The numerical simulation is performed with $\Delta t = 0.005$ for 200 time steps.

Numerical results In this experiment, our primary focus is on evaluating the efficiency of the dynamically load-balanced p-adaptive solver rather than investigating physical properties. In other words, the characteristic convective time is not concerned for the physical duration of computation. However, we use reference data from https://cfd.ku.edu/hiocfd/case_c3.5.pdf to compare temporal evaluations of kinetic energy and enstrophy with respect to spatial discretization. Fig. 4.2.1 displays that the solutions of uniform DG and ADG schemes exhibit similar accuracy in terms of physical quantities and Fig. 4.2.2 describes similar contour slices of density. On the other hand, as shown in Table 4.2.1, employing the p-adaptive algorithms with dynamic load balancing leads to significant runtime reductions, up to 86.8% compared to the DG(4) solver. As outlined in the graph, e.g., Fig. 4.2.3, using the SD indicator only generates $P4$ elements, requiring more CPU run time than other indicators. As observed earlier, the usage of the PP indicator does not result in much refinement of polynomial orders.

We explore the numerical performance of ADG solvers with the SD indicator, aiming to compare outcomes based on different mesh partitioners. Employing a two-level mesh partitioning approach, each node is constructed with 12P × 4T cores. By varying the number of cores from 768 to 1,536, we can observe the CPU runtime for each mesh partitioner in Fig. 4.2.4. Among the mesh partitioners, GeMPa as the coarse level partitioner exhibits the best results in both MPI mode and the hybrid approaches. However, GeMPa combined with METIS or Zoltan shows comparable performance to ParMETIS and Zoltan in MPI mode. On the other hand, using Zoltan for the distributed memory level results in less efficiency compared to ParMETIS and GeMPa. In the shared memory level, there is no significant difference in runtime between METIS and Zoltan. Additionally, we evaluate the numerical scalability in Table 4.2.2. Higher scalability results are observed for ParMETIS and GeMPa as the coarse-level distributors compared to MPI mode. Conversely, employing Zoltan for the coarse level results in poorer numerical performance and scalability.

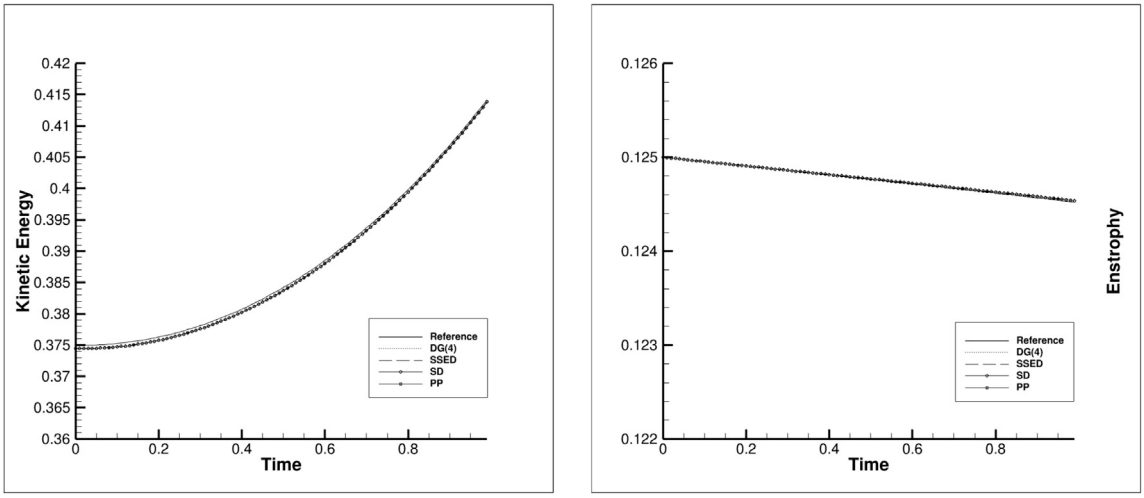


Fig. 4.2.1. The temporal evolution of kinetic energy (left) and enstrophy (right).

Table 4.2.1
CPU run time of simulations with 960 cores which load-balanced by ParMETIS in MPI mode.

	Total nonlinear It.	Total linear It.	Run time(h)	Reduction(%)
DG(4)	1792	4782	13.18	
ADG-SSED	1716	4322	3.15	76.1
ADG-SD	1711	4527	6.10	53.7
ADG-PP	1713	4571	1.75	86.8

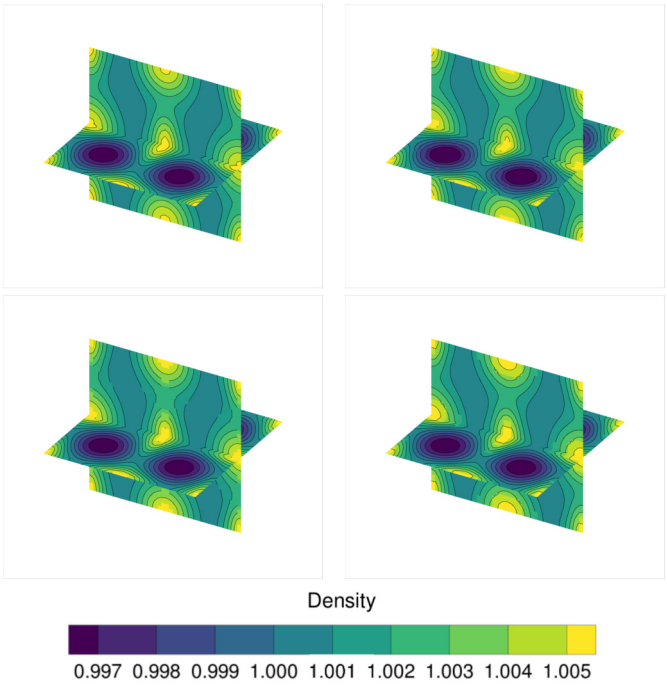


Fig. 4.2.2. Contour slice plots of density at $t = 1$; DG(4) (top left), ADG-SSED (top right), ADG-SD (bottom left) and ADG-PP (bottom right).

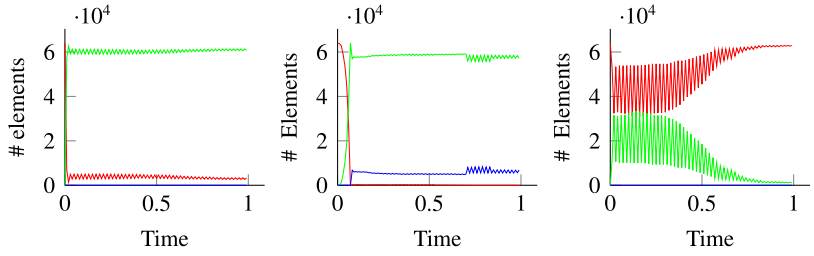


Fig. 4.2.3. Change of polynomial orders in time with SSED (left), SD (center), and PP (right) indicators, which are colored by red for $P2$, green for $P3$, and blue for $P4$, respectively (x-axis: time domain and y-axis: the number of elements).

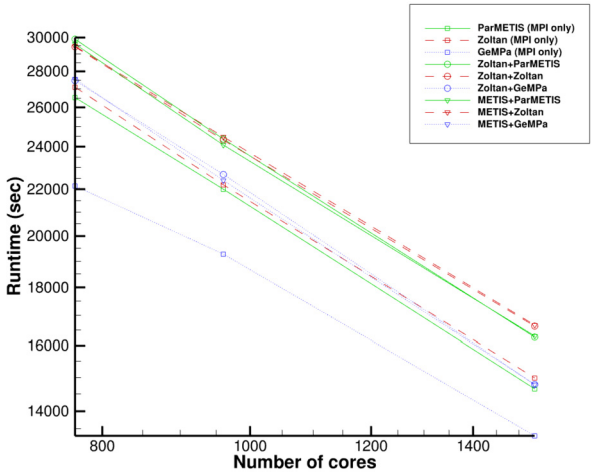


Fig. 4.2.4. Comparison results of run time with respect to mesh partitioning: MPI-only (□) or two-level partitioning (○ by Zoltan and ▽ by METIS for the fine level, respectively).

Table 4.2.2
Scalability rates for ADG-SD solver simulations. MPI mode: 48P per node, hybrid: 12P × 4T per node with up to 1536 cores.

Coarse level	Fine level	Scalability rate
ParMETIS	(MPI only)	0.906
	METIS	0.908
	Zoltan	0.918
Zoltan	(MPI only)	0.904
	METIS	0.884
	Zoltan	0.885
GeMPa	(MPI only)	0.832
	METIS	0.932
	Zoltan	0.929

Remark. With the structured Cartesian mesh in the TGV simulation, it is observed that mesh partitioning does not affect the convergence results. Specifically, the total number of linear/nonlinear iterations required for convergence does not depend on the choice of mesh partitioners in either MPI mode or hybrid mode, regardless of the number of cores. However, in practical terms, different mesh decompositions can lead to variations in numerical convergence. For instance, the numerical performance of the linear solver with block preconditioning is influenced by the computational mesh decomposition. Consequently, the numerical convergence, concerning both linear and nonlinear iterations, may vary based on the chosen method of mesh partitioning.

4.3. Periodic laminar flow past a cylinder

Following the classical numerical test of the well-known vortex shedding phenomenon, we examine the unsteady flow around a circular

Table 4.3.1
Average drag coefficient and root mean square of the lift coefficient with reference results.

	$\bar{C}_D$.	C'_L	Ma	Domain size	Run time (hh:mm)
Naddei et al. [9]	1.33	0.227	0.1	$200D$	
Ferrero et al. [37]	1.34	-	0.2	$100D$	
Rajani et al. [38]	1.34	0.179	0	$20D$	
Williamson (exp.) [39]	1.33	-			
Tritton (exp.) [40]	1.26	-			
Norberg (exp.) [41]	-	0.227			
ADG-SSED	1.35	0.229	0.2	$100D$	10:39
ADG-SD	1.34	0.228	0.2	$100D$	08:08
ADG-PP	1.35	0.229	0.2	$100D$	05:01

Table 4.3.2
Numerical performance of ADG-SD solver simulations with respect to mesh partitioning (MPI mode: 48P per node, hybrid: 12P × 4T per node).

Coarse	Fine	# cores	Nonlin./lin. It.	Run time(s) [Scalability rate]
ParMETIS	(MPI only)	480	22,724 / 374,716	29,309
		960	22,726 / 376,144	16,186 [0.90]
	METIS	480	22,776 / 369,909	30,257
		960	22,745 / 371,833	16,853 [0.90]
	Zoltan	480	22,769 / 363,927	29,129
		960	22,761 / 380,201	18,252 [0.80]
Zoltan	(MPI only)	480	22,750 / 374,660	29,953
		960	22,766 / 374,193	16,945 [0.88]
	METIS	480	22,749 / 373,797	31,204
		960	22,771 / 370,777	17,274 [0.90]
	Zoltan	480	22,778 / 362,152	28,857
		960	22,756 / 367,430	16,907 [0.85]
GeMPa	(MPI only)	480	22,798 / 372,875	29,897
		960	22,751 / 372,195	20,050 [0.75]
	METIS	480	22,765 / 364,299	29,045
		960	22,774 / 364,193	17,255 [0.84]
	Zoltan	480	22,758 / 372,452	31,239
		960	22,752 / 369,810	16,954 [0.92]

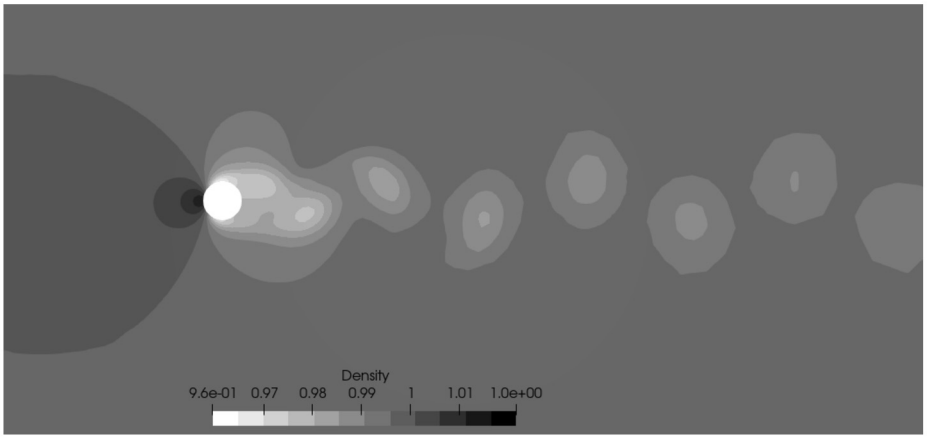


Fig. 4.3.1. Laminar flow past a cylinder where $Re = 100$, $Ma = 0.2$ and $t = 500$.

cylinder with $Re = 100$ and $Ma = 0.2$. Characteristic far-field boundary conditions are applied to a circular boundary, located at a distance equal to 100 times the cylinder diameter D. The solid wall is treated as adiabatic. The 3D domain is discretized using an unstructured O-mesh consisting of 3,958 prism elements and 7,680 hexahedral elements. The mesh is extruded based on a 2D structure. Several simulations were conducted with the static DG scheme to assess the behavior of the proposed discretization in the presence of unsteady flow fields, with computations continuing until periodicity is achieved with $\Delta t = 0.05$. For additional numerical solutions using different types of discretization, we refer to [37].

Numerical results After performing the simulation with the uniform DG scheme for $t \in [0, 500]$, the periodicity is observed where at least 6 shedding cycles appear. With the solution state for $t = 500$ as the initial state illustrated in Fig. 4.3.1, we simulate 1,000 time steps with applying the p-adaptive algorithm. To verify the characteristic of the vortex shedding, we compute the average drag coefficient ($\bar{C}_D$) of a shedding cycle and the root mean square of the lift coefficient (C'_L). We then compare our result with other numerical and experimental results in Table 4.3.1.

As seen in Fig. 4.3.2, higher refinements are derived by the SSED indicator. In a similar way to the previous numerical examples, the PP indicator imposes lower order simulations. On the other hand, we can observe high-order polynomial elements near the boundary of the cylinder and flow paths past the cylinder in Fig. 4.3.3 for all indicators.

Table 4.3.2 describes the numerical performance of the ADG-SD solver with various dynamic load balancers in MPI mode or hybrid mode. While as remarked in Section 4.2, the mesh partitioning does

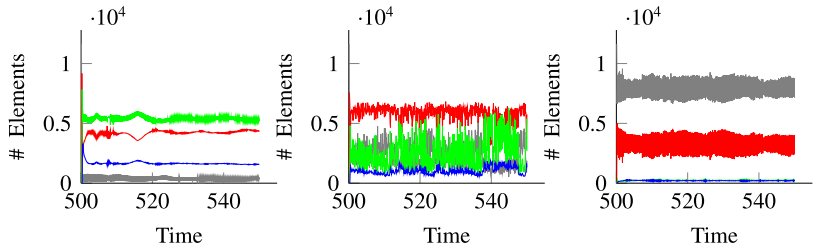


Fig. 4.3.2. Change of polynomial orders in time with SSED (left), SD (center), and PP (right) indicators, which are colored by gray for $P1$, red for $P2$, green for $P3$, and blue for $P4$, respectively (x-axis: time domain and y-axis: the number of elements).

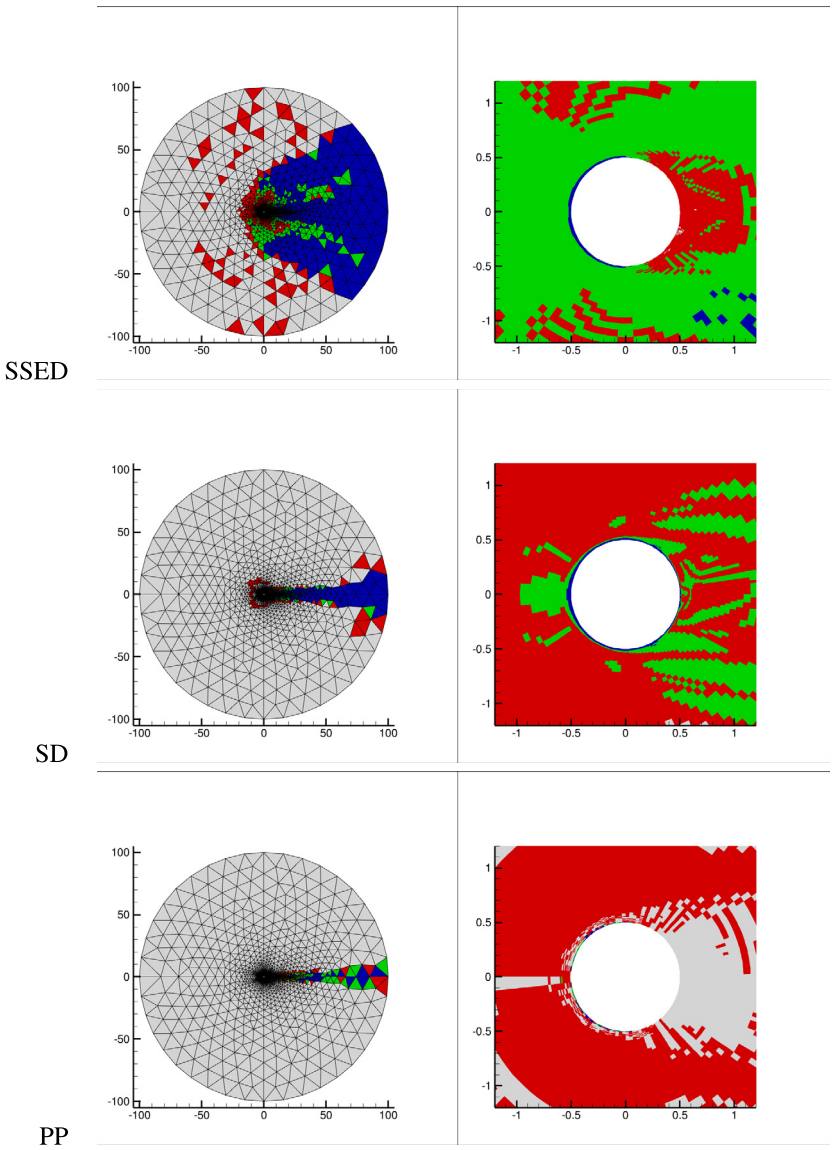


Fig. 4.3.3. Local polynomial distribution at $t = 550$ in the global view (left) and the close view near the cylinder (right) indicated by colors e.g., gray for $P1$, red for $P2$, green for $P3$ and blue for $P4$, respectively.

not have any significant impact on the total number of nonlinear/linear iterations for the structured mesh, the different choices of distributors lead to distinct qualities of linear solvers for the unstructured mesh. Since local block preconditioning relies on mesh decomposition, in the unstructured mesh domain, the numerical performance exhibits a dependency on mesh distribution and the number of cores for parallel computation. On the other hand, the numerical results exhibit better scalability by two-level partitioning. In particular, when we combine SFC based partitioning for distributed memory level and hyper-graph based partitioning for shared memory level, we can observe significant improvement in scalability. On the other hand, the hypergraph partitioning at the fine level failed to show the benefit of the two-level decomposition. For instance, with ParMETIS/Zoltan (MPI level) + Zoltan (thread level), the linear iterations to be performed are significantly increased in the number of cores, which implies poorer performance of linear solvers.

5. Conclusion

In our investigation, we conducted a comprehensive analysis of *p*-adaptive DG solvers to assess their effectiveness in simulating various fluid flow phenomena. We compared the performance of *p*-adaptive schemes with uniform DG approaches across different scenarios, including vortex transportation, unsteady Taylor-Green vortex flow, and laminar flow past a cylinder. Our findings consistently underscored the superiority of *p*-adaptive strategies, particularly those utilizing *a posteriori* error indicators. These methods not only exhibited superior solution accuracy but also demonstrated remarkable computational efficiency gains across a diverse range of flow scenarios.

Smoothness-based indicators adjusted local polynomial degrees, reducing runtime significantly by up to 90% without compromising accuracy in physical quantities or flow characteristics. Specifically, while the PP indicator led to relatively passive refinement, the numerical solutions showed comparable convergence. The SSED and SD indicators resulted in similar distributions of polynomial degree on structured meshes. Additionally, we observed significantly enhanced efficiency through dynamic load balancing compared to uniform mesh distributions. Utilizing multi-threading, numerical performance achieved better scalability than in pure MPI mode. Notably, the combination of the SFC-based partitioner GeMPa for the coarse level with graph-based partitioning for the fine level significantly enhanced numerical scalability.

Although the choice of domain decomposition methods had no significant impact on numerical performance for structured mesh problems, the quality of linear solvers depended on the type of mesh partitioning for unstructured meshes. For instance, employing ParMETIS or Zoltan for MPI level and Zoltan for thread level resulted in poor performance in scalability, suggesting that hypergraph mesh partitioning may fail to define sufficiently good local preconditioners for unstructured cases. In our proposed method, the quantity of partitioning weights is determined by the degree of freedoms relative to the polynomial degree, making it suitable for load balancing when solving linear systems. However, it may not be optimal for computing residual vectors, defining Jacobian matrices, and discretizing problems, as these processes heavily rely on the number of quadrature points for numerical integration and other discretization parameters. Therefore, we plan to investigate task-based load balancing for unsteady problems in our future studies, as demonstrated in [34]. It is also planned to address the topic of dynamic *hp* adaptation at runtime. Static *hp* adaptation techniques have been already developed in CODA and tested for a number of applications [42]. However, *hp* dynamic adaptation comes with a number of technical challenges that need to be addressed, including efficient parallel remeshing, conservative interpolation from an arbitrary mesh to another with *p* heterogeneity, and efficient dynamic mesh partitioning. These points have not been treated and will be the topic for future research.

CRediT authorship contribution statement

Yongseok Jang: Investigation, Software, Visualization, Writing – original draft, Writing – review & editing. **Emeric Martin:** Writing – review & editing. **Jean-Baptiste Chapelier:** Formal analysis, Writing – review & editing. **Vincent Couaillier:** Formal analysis, Project administration, Supervision.

Acknowledgements

The work presented here has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No. 956104 ("NextSim"). We gratefully acknowledge this support, which has been instrumental in conducting the research described in this paper.

Data availability

Data will be made available on request.

References

[1] F. Bassi, S. Rebay, A high-order accurate discontinuous finite element method for the numerical solution of the compressible Navier–Stokes equations, J. Comput. Phys. 131 (2) (1997) 267–279.
[2] F. Bassi, S. Rebay, GMRES discontinuous Galerkin solution of the compressible Navier-Stokes equations, in: Discontinuous Galerkin Methods: Theory, Computation and Applications, Springer, 2000, pp. 197–208.
[3] F. Bassi, L. Botti, A. Colombo, D.A. Di Pietro, P. Tesini, On the flexibility of agglomeration based physical space discontinuous Galerkin discretizations, J. Comput. Phys. 231 (1) (2012) 45–65.
[4] I. Babuška, M. Suri, The *p* and *hp* versions of the finite element method, basic principles and properties, SIAM Rev. 36 (4) (1994) 578–632.
[5] P. Houston, E. Süli, *hp*-adaptive discontinuous Galerkin finite element methods for first-order hyperbolic problems, SIAM J. Sci. Comput. 23 (4) (2001) 1226–1252.
[6] P. Houston, E. Süli, A note on the design of *hp*-adaptive finite element methods for elliptic partial differential equations, Comput. Methods Appl. Mech. Eng. 194 (2–5) (2005) 229–243.
[7] R. Hartmann, P. Houston, Adaptive discontinuous Galerkin finite element methods for the compressible Euler equations, J. Comput. Phys. 183 (2) (2002) 508–532.
[8] G. Gassner, C. Altmann, F. Hindenlang, M. Staudenmeier, C. Munz, Explicit discontinuous Galerkin schemes with adaptation in space and time, in: 36th CFD/ADIGMA Course on *hp*-Adaptive and *hp*-Multigrid Methods, 2009, pp. 1–68.
[9] F. Naddei, M. de la Llave Plata, V. Couaillier, F. Coquel, A comparison of refinement indicators for *p*-adaptive simulations of steady and unsteady flows using discontinuous Galerkin methods, J. Comput. Phys. 376 (2019) 508–533.
[10] 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, Comput. Fluids 139 (2016) 36–46.
[11] L. Wang, D.J. Mavriplis, Adjoint-based *hp* adaptive discontinuous Galerkin methods for the 2D compressible Euler equations, J. Comput. Phys. 228 (20) (2009) 7643–7661.
[12] K.J. Fidkowski, Output-based space–time mesh optimization for unsteady flows using continuous-in-time adjoints, J. Comput. Phys. 341 (2017) 258–277.
[13] P.-O. Persson, J. Peraire, Sub-cell shock capturing for discontinuous Galerkin methods, in: 44th AIAA Aerospace Sciences Meeting and Exhibit, 2006, p. 112.
[14] G. Kuru, M. de la Llave Plata, V. Couaillier, R. Abgrall, F. Coquel, An adaptive variational multiscale discontinuous Galerkin method for large eddy simulation, in: 54th AIAA Aerospace Sciences Meeting, 2016, p. 0584.
[15] K.J. Fidkowski, D.L. Darmofal, Review of output-based error estimation and mesh adaptation in computational fluid dynamics, AIAA J. 49 (4) (2011) 673–694.
[16] W. Li, A.K. Pandare, H. Luo, J. Bakosi, J. Waltz, A parallel *p*-adaptive discontinuous Galerkin method for the Euler equations with dynamic load-balancing on tetrahedral grids, Int. J. Numer. Methods Fluids 95 (12) (2023) 1913–1932.
[17] J. Jägersküpper, D. Vollmer, On highly scalable 2-level-parallel unstructured CFD, in: ECCOMAS Congress 2022, 2022, pp. 1–12.
[18] T. Leicht, J. Jägersküpper, D. Vollmer, A. Schwöppe, R. Hartmann, J. Fiedler, T. Schlauch, DLR-Project Digital-X - next generation CFD solver 'Flucs', in: Deutscher Luft- und Raumfahrtkongress 2016, 2016, pp. 1–14, https://elib.dlr.de/111205/.
[19] P. Stefanin Volpiani, J.-B. Chapelier, A. Schwöppe, J. Jägersküpper, S. Champagneux, Aircraft simulations using the new CFD software from ONERA, DLR, and Airbus, J. Aircr. 61 (3) (2024) 1–13.
[20] G. Karypis, K. Schloegel, V. Kumar, ParMETIS: Parallel graph partitioning and sparse matrix ordering library, Tech. rep, University of Minnesota, 1997, https://conservancy.umn.edu/server/api/core/bitstreams/1a922221-2cb8-4726-b56a-7cc3c88d0d18/content.
[21] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, C. Vaughan, Zoltan data management services for parallel dynamic applications, Comput. Sci. Eng. 4 (2) (2002) 90–96.
[22] R. Borrell, J.C. Cajas, D. Mira, A. Taha, S. Koric, M. Vázquez, G. Houzeaux, Parallel mesh partitioning based on space filling curves, Comput. Fluids 173 (2018) 264–272.
[23] P.L. Roe, Approximate Riemann solvers, parameter vectors, and difference schemes, J. Comput. Phys. 43 (2) (1981) 357–372.
[24] P.L. Roe, Characteristic-based schemes for the Euler equations, Annu. Rev. Fluid Mech. 18 (1) (1986) 337–365.
[25] T. Houba, A. Dasgupta, S. Gopalakrishnan, R. Gosse, S. Roy, Supersonic turbulent flow simulation using a scalable parallel modal discontinuous Galerkin numerical method, Sci. Rep. 9 (1) (2019) 14442.
[26] H. Bijl, M.H. Carpenter, V.N. Vatsa, C.A. Kennedy, Implicit time integration schemes for the unsteady compressible Navier–Stokes equations: laminar flow, J. Comput. Phys. 179 (1) (2002) 313–329.
[27] C.A. Kennedy, M.H. Carpenter, Diagonally implicit Runge-Kutta methods for ordinary differential equations. A review, Tech. rep., NASA, 2016, https://ntrs.nasa.gov/api/citations/20160005923/downloads/20160005923.pdf.
[28] W.A. Mulder, B. Van Leer, Experiments with implicit upwind methods for the Euler equations, J. Comput. Phys. 59 (2) (1985) 232–246.

[29] T.T. Chisholm, D.W. Zingg, A Jacobian-free Newton–Krylov algorithm for compressible turbulent fluid flows, J. Comput. Phys. 228 (9) (2009) 3490–3507.
[30] P.D. Hovland, L.C. McInnes, Parallel simulation of compressible flow using automatic differentiation and PETSc, Parallel Comput. 27 (4) (2001) 503–519.
[31] R. Alexander, Diagonally implicit Runge–Kutta methods for stiff ODE's, SIAM J. Numer. Anal. 14 (6) (1977) 1006–1021.
[32] J.G. Bautista, M. de la Llave Plata, V. Couaillier, M. Visonneau, K. Schneider, *h*-adaptation for high-order discontinuous Galerkin schemes built on local multiwavelet analysis, Comput. Fluids 256 (2023) 105844.
[33] R. Borrell, G. Oyarzun, D. Dosimont, G. Houzeaux, Parallel SFC-based mesh partitioning and load balancing, in: 2019 IEEE/ACM 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA), 2019, pp. 72–78.
[34] G. Baldan, R. Borell, J. Jägersküpper, A runtime-based dynamic mesh-partitioning approach, in: 8th European Congress on Computational Methods in Applied Sciences and Engineering, ECCOMAS Congress 2022, Scipedia, 2022, pp. 1–12.
[35] S. Rajamanickam, E.G. Boman, An evaluation of the Zoltan parallel graph and hypergraph partitioners, Tech. rep., Sandia National Lab. (SNL-NM), Albuquerque, NM (United States), 2011, https://www.osti.gov/servlets/purl/1111609.
[36] J. Kok, A high-order low-dispersion symmetry-preserving finite-volume method for compressible flow on curvilinear grids, J. Comput. Phys. 228 (18) (2009) 6811–6832.
[37] A. Ferrero, F. Larocca, G. Puppo, A robust and adaptive recovery-based discontinuous Galerkin method for the numerical solution of convection–diffusion equations, Int. J. Numer. Methods Fluids 77 (2) (2015) 63–91.
[38] B. Rajani, A. Kandasamy, S. Majumdar, Numerical simulation of laminar flow past a circular cylinder, Appl. Math. Model. 33 (3) (2009) 1228–1247.
[39] C.H. Williamson, Oblique and parallel modes of vortex shedding in the wake of a circular cylinder at low Reynolds numbers, J. Fluid Mech. 206 (1989) 579–627.
[40] D.J. Tritton, Experiments on the flow past a circular cylinder at low Reynolds numbers, J. Fluid Mech. 6 (4) (1959) 547–567.
[41] C. Norberg, Fluctuating lift on a circular cylinder: review and new measurements, J. Fluids Struct. 17 (1) (2003) 57–96.
[42] J.-B. Chapelier, F. Basile, F. Naddei, M. de la Llave Plata, V. Couaillier, R. Laraufie, *hp* adaptive discontinuous Galerkin strategies driven by a posteriori error estimation with application to aeronautical flow problems, Adv. Appl. Mech. 58 (2024).