Comparison of algorithm in Aerosol and Aghora for compressible flows

Dragan Mbengoue, Damien Genet, Cedric Lachat, Emeric Martin, Maxime Mogé, Vincent Perrier, Florent Renac, François Rue, Mario Ricchiuto

Comparison of algorithm in Aerosol and Aghora for compressible flows

D.A. Mbengoue, D. Genet, C. Lachat, E. Martin, M. Mogé, V. Perrier, F. Renac, F. Rué, M. Ricchiuto

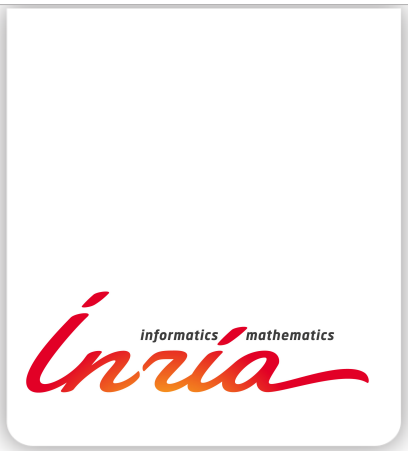
RESEARCH REPORT

N° 8200

January 2013

Project-Teams BACCHUS and CAGIRE

ISSN 0249-6399 ISRN INRIA/RR--8200--FR+ENG



Comparison of algorithm in Aerosol and Aghora for compressible flows

D.A. Mbengoue*, D. Genet†, C. Lachat‡, E. Martin§, M. Mogé¶, V. Perrier‖, F. Renac**, F. Rué††, M. Ricchiuto‡‡

Project-Teams BACCHUS and CAGIRE

Research Report n° 8200 — January 2013 — 16 pages

Abstract: This article summarizes the work done within the COLARGOL project during CEMRACS 2012. The aim of this project is to compare the implementations of high order finite elements methods for compressible flows that have been developed at ONERA and at INRIA for about one year, within the AGHORA and AEROSOL libraries.

Key-words: Finite elements, parallel performance, compressible flows

* INRIA Bordeaux Sud-Ouest, Equipe BACCHUS
† INRIA Bordeaux Sud-Ouest, Equipe BACCHUS
‡ INRIA Bordeaux Sud-Ouest, Equipe BACCHUS
§ ONERA
¶ LMAP, Université de Pau et des Pays de l'Adour
‖ LMAP, Université de Pau et des Pays de l'Adour, INRIA Bordeaux Sud-Ouest, Equipe CAGIRE
** ONERA
†† INRIA, service SED
‡‡ INRIA Bordeaux Sud-Ouest, Equipe BACCHUS

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

351, Cours de la Libération
Bâtiment A 29
33405 Talence Cedex

Comparison of algorithm in Aerosol and Aghora for compressible flows

Résumé : Cet article résume le travail effectué durant le projet COLARGOL pendant le CEMRACS 2012. Le but de ce projet est de comparer les implémentations de méthodes d'éléments finis d'ordre élevé pour les fluides compressibles devéloppées à l'ONERA et à l'INRIA depuis environ un an dans les bibliothèques AGHORA et AEROSOL.

Mots-clés : Finite elements, parallel performance, compressible flows

Contents

Introduction

On the need for high order in aerodynamics

Over the last three decades, most of the simulations for designing aircraft have been reduced to second order finite volumes methods, mainly with RANS (Reynolds Averaged Navier-Stokes) models for turbulence. The usual trick for having an accurate solution with a RANS simulation consists in performing a sequence of computations on more and more refined meshes. However, the RANS approach can be inefficient for simulating some problems that are intrisically time dependent, because it is a time averaged model, and thus stationary. For time dependent simulations, the most accurate method is called DNS (Direct Numerical Simulation) and consists in meshing the domain at the turbulent scale, without any turbulence model. However, due to the required mesh fineness, DNS is often considered only for low Reynolds numbers (since the number of cells increases as $\mathrm{Re}^{9/4}$) and/or for academic configurations, for which efficient finite differences methods can be used. An intermediate solution, between DNS and RANS approach consists in applying a spatial filter to the Navier-Stokes equations. The resulting model is time dependent, but shall be closed for taking into account small scales. This method is called the Large Eddy Simulation (LES) approach. For both DNS and LES, the accuracy yielded by adaptive mesh refinement is harder to implement because the problem is time dependent. It would therefore imply theoretically to perform a mesh adaptation at each time step, which is very costly. It also raises issues regarding dynamic load balancing as far as parallel computing is concerned.

How to derive a high order method

Mathematically speaking, a sufficient condition for having a high order approximation of a function is to find a piecewise polynomial approximation of this function, for

example by interpolation or L^2-projection. Once the type of approximation is chosen, a numerical scheme must be derived from this approximation. The following methods can be considered.

- **High order finite volume methods**
  In finite volume approximations, the solution is considered as piecewise constant. For getting a high order method, a high order polynomial representation can be computed, by interpolating the values on the neighboring cells. For example, in one dimension, a second order polynomial can be computed on one cell by interpolating the values on the left and right cells. This can be generalized to higher dimensions and to higher order polynomial approximations, and the price to pay for having a high order representation is to visit more and more neighbors. Moreover, in a hyperbolic framework, the aim is to rebuild a *non oscillatory* interpolation [**?**], which complicates the problem. This method is not compact, and therefore is not well suited to parallel computing.

- **Continuous Galerkin method**
  In this method, the solution is approximated by piecewise continuous polynomial functions. The numerical scheme is then obtained by writing the L^2 orthogonality between the approximation basis and the equation projected on the approximation space. It is well adapted to some problems such as purely parabolic or elliptic problems. For Euler equations, and more generally for hyperbolic systems (even linear), this method is known to be unstable. This method can be stabilized, for example with the SUPG method [**?**]. Nevertheless, this method depends on parameters that can be hard to tune.

- **Residual Distribution schemes**
  The development of fluctuation schemes began with the early work of Phil Roe [**?**]. Residual distribution is a weighted residual approach in which local discrete equations are derived as a sum of elemental contributions proportional to element integrals of the equations (the cell residuals) via matrix weights. The basics of the method are thoroughly discussed in [**?**]. As in continuous and discontinuous finite element methods, the key toward high accuracy is the use of a high order polynomial representation of the unknowns in the computation of the cell residuals [**?**]. Although this approach has shown great potential in steady applications [**?**, **?**], its current state of the art [**?**] shows that further work is needed to bring the method to the level of maturity of more popular techniques, such as Discontinuous Galerkin.

- **Discontinuous Galerkin method**
  The development of discontinuous Galerkin for nonlinear hyperbolic equations began in [**?**]. Its stabilization for flows with shocks was developed in the 90's, mainly by Cockburn and Shu (see [**?**] for a review). In the same time, a solution for dealing with Navier-Stokes equations was proposed in [**?**]. In spite of its cost (it has much more degrees of freedom than classical continuous finite elements methods), it is attractive because it is naturally L^2 stable for linear problems, because a cell entropy inequality can be proven [**?**], and also because it has a compact stencil so that it has a good behavior in parallel environment [**?**]. The success of these methods lies in their flexibility thanks to their high degree of locality. These properties make the DG method well suited to parallel computing, hp-refinement, hp-multigrid, unstructured meshes, the weak application of boundary conditions, etc.

The development of different high order methods for aerospace application was the topic of the European project ADIGMA, and we refer to [**?**] for recent developments on this topic.

1 Discontinuous Galerkin methods and libraries

1.1 The Euler model

Let $\Omega \subset \mathbb{R}^d$ be a bounded domain where d is the space dimension and consider the following problem

$$\partial_t \mathbf{u} + \nabla \cdot \mathbf{f}(\mathbf{u}) = 0, \quad \text{in } \Omega \times (0, \infty) \ , \tag{1}$$

with initial condition $\mathbf{u}(\cdot, 0) = \mathbf{u}_0(\cdot)$ in Ω and appropriate boundary conditions prescribed on $\partial\Omega$. The vector $\mathbf{u} = (\rho, \rho\mathbf{v}, \rho E)^\top$ represents the conservative variables with ρ the density, $\mathbf{v} \in \mathbb{R}^d$ the velocity vector and $E = \varepsilon(\mathrm{p}, \rho) + \mathbf{v}^2/2$ where ε is the specific internal energy. Here, we suppose that the fluid follows the perfect gas equation

$$\varepsilon(\mathrm{p}, \rho) = \frac{\mathrm{p}}{(\gamma - 1)\rho} \ ,$$

where p denotes the pressure and γ is the ratio of specific heats. The nonlinear convective fluxes in (1) are defined by

$$\mathbf{f}(\mathbf{u}) = \begin{pmatrix} \rho\mathbf{v}^\top \\ \rho\mathbf{v}\mathbf{v}^\top + \mathrm{p}\mathbf{I} \\ (\rho E + \mathrm{p})\mathbf{v}^\top \end{pmatrix} \ . \tag{2}$$

The problem (1) is hyperbolic provided the conservative variable vector takes values in the set of admissible states

$$\Psi = \{\mathbf{u} \in \mathbb{R}^{d+2} : \ \rho > 0, \ E - \frac{\mathbf{v}^2}{2} > 0\} \ . \tag{3}$$

1.2 Runge-Kutta discontinuous Galerkin formulation

The discontinuous Galerkin method is a finite element method in which the weak formulation is projected on a space of discontinuous piecewise polynomials of the problem (1). The domain Ω is partitioned into a shape-regular mesh Ω_h consisting of nonoverlapping and nonempty elements κ of characteristic size $h := \min\{h_\kappa, \kappa \in \Omega_h\}$ where h_κ is a d-dimensional measure of κ. We define the sets $\mathcal{E}_i$ and $\mathcal{E}_b$ of interior and boundary faces in Ω_h, respectively.

We look for approximate solutions in the function space of discontinuous polynomials $\mathcal{V}_h^p = \{\phi \in L^2(\Omega_h) : \ \phi|_\kappa \circ F_\kappa^{-1} \in \mathcal{P}^p(\hat{\kappa}), \ \forall \kappa \in \Omega_h\}$, where $\mathcal{P}^p(\hat{\kappa})$ denotes the polynomial space associated to the reference element $\hat{\kappa}$ corresponding to the element κ. Each physical element κ is the image of one of the following reference shapes $\hat{\kappa}$ through the mapping F_κ: simplex (line, triangle or tetrahedron), tensor elements (quadrangle, hexaedron), prism or pyramid. The space $\mathcal{P}^p(\hat{\kappa})$ might be composed of the set of polynomials with degree lower or equal to p in the case of simplex, or of tensor product of one dimensional polynomials with degree lower or equal to p in the case of tensor elements, or of a tensor of one dimensional basis and two dimensional

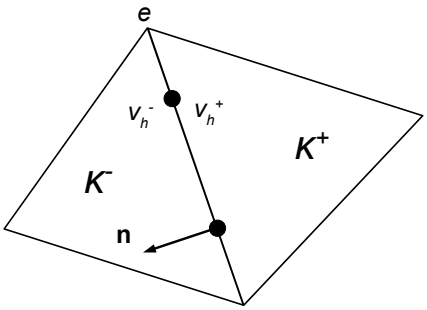


Figure 1: Inner and exterior elements κ^+ and κ^- and definition of traces $v_h^{\pm}$ on the interface e and of the unit outward normal vector $\mathbf{n}$.

basis on a triangle if $\hat{\kappa}$ is a prism, or a conical product of one dimensional basis and two dimensional basis on a quadrangle if $\hat{\kappa}$ is a pyramid. The numerical solution of equation (1) is sought under the form

$$\mathbf{u}_h(\mathbf{x},t) = \sum_{l=1}^{N_\kappa} \phi_\kappa^l(\mathbf{x})\mathbf{U}_\kappa^l(t), \quad \forall \mathbf{x} \in \kappa,\, \kappa \in \Omega_h,\, \forall t \geq 0 \ , \tag{4}$$

where $(\mathbf{U}_\kappa^l)_{1\leq l\leq N_\kappa}$ are the degrees of freedom in the element κ. The semi-discrete form of equation (1) reads: find $\mathbf{u}_h$ in $[\mathcal{V}_h^p]^{d+2}$ such that for all v_h in $\mathcal{V}_h^p$ we have

$$\int_{\Omega_h} v_h \partial_t \mathbf{u}_h d\mathbf{x} + \mathcal{B}_h(\mathbf{u}_h, v_h) = 0 \ . \tag{5}$$

Hereafter, we will use the notation $[\![\phi]\!] = \phi^+ - \phi^-$ which denotes the jump operator defined for a given interface $e \in \mathcal{E}_i$. Here, ϕ^+ and ϕ^- are the traces of any quantity ϕ on the interface e taken from within the interior of the element κ^+ and the interior of the neighboring element κ^-, respectively (see Figure 1).

The space discretization in equation (5) can then be written as

$$\mathcal{M}_\kappa \left(\frac{\mathbf{u}_h^{n+1} - \mathbf{u}_h^n}{\delta t} \right) + \mathcal{B}_h(\mathbf{u}_h^n, v_h) = 0 \ . \tag{6}$$

for the explicit Euler time stepping, the extension to other explicit time stepping being straightforward. In (6) $\mathcal{M}_\kappa$ denotes the local mass matrix defined as

$$\forall (i,j) \in [0; N_\kappa] \times [0; N_\kappa] \qquad \mathcal{M}_\kappa^{(i,j)} = \int_\kappa \phi_\kappa^i(\mathbf{x})\phi_\kappa^j(\mathbf{x})\, \mathrm{d}\mathbf{x} \ ,$$

and the space discretization operator $\mathcal{B}_h$ is defined by

$$\begin{aligned} \mathcal{B}_h(\mathbf{u}_h, v_h) \quad = \quad & -\int_{\Omega_h} \mathbf{f}(\mathbf{u}_h) \cdot \nabla v_h dV \\ & + \int_{\mathcal{E}_i} [\![v_h]\!] \tilde{\mathbf{f}}(\mathbf{u}_h^+, \mathbf{u}_h^-, \mathbf{n}) dS \\ & + \int_{\mathcal{E}_b} v_h^+ \mathbf{f}(\mathbf{u}_b(\mathbf{u}_h^+, \mathbf{n})) \cdot \mathbf{n} dS \ , \end{aligned} \tag{7}$$

where $\mathbf{n}$ denotes the unit outward normal vector to an element κ^+ (see Figure 1) and $\mathbf{u}_b$ is an appropriate operator which allows to impose boundary conditions on $\mathcal{E}_b$. The numerical flux $\tilde{\mathbf{f}}$ may be chosen to be any monotonic Lipschitz function satisfying consistency, conservativity and entropy dissipativity properties (see [**?**] for instance).

Throughout this study, the semi-discrete equation (5) is advanced in time by means of an explicit treatment with third-order strong stability preserving Runge-Kutta methods [**?**, **?**].

1.3 Algorithm

As an explicit Runge-Kutta time stepping was chosen, the algorithm consists in computing the spatial residual, and then in updating the intermediate (or the final) steps of the Runge-Kutta method by inverting the mass matrix. In the discontinuous Galerkin method, the mass matrix is block diagonal. It can actually be diagonal provided an orthogonal basis is used on each element. We point out that this step of the method does not require any communication in a distributed memory environment if all elements are on the same process (which is always the case). We now dwell on the different loops that must be made for computing the local spatial residual defined on (7). We detail the computations needed for the elements loop, the other loops being similar.

We are interested in computing

$$\int_\kappa \mathbf{f}(\mathbf{u}_h)\cdot\nabla v_h dV$$

for all basis function v_h of κ. Using the definition of the basis given in Section 1.2

$$\int_\kappa \mathbf{f}(\mathbf{u}_h)\cdot\nabla\phi^i_\kappa dV = \int_\kappa \mathbf{f}(\textstyle\sum_{l=1}^{N_\kappa}\phi^l_\kappa(\mathbf{x})\mathbf{U}^l_\kappa)\cdot\nabla\phi^i_\kappa dV \ .$$

As in Section 1.2, we denote by F_κ the map from κ to $\hat\kappa$. In the integral, we change the variable $\mathbf{x}$ by $\mathbf{x}=F_\kappa(\hat{\mathbf{x}})$, so that

$$\begin{aligned}\int_\kappa \mathbf{f}\left(\textstyle\sum_{l=1}^{N_\kappa}\phi^l_\kappa(\mathbf{x})\mathbf{U}^l_\kappa\right)\cdot\nabla\phi^i_\kappa dV &= \int_{\hat\kappa}\mathbf{f}\left(\textstyle\sum_{l=1}^{N_\kappa}\phi^l_\kappa\circ F_\kappa(\hat{\mathbf{x}})\mathbf{U}^l_\kappa\right)\cdot DF_\kappa^{-1}\nabla\hat\phi^i_{\hat\kappa}\,|\det DF_\kappa|\ \mathrm{d}\hat{\mathbf{x}}\\ &= \int_{\hat\kappa}\mathbf{f}\left(\textstyle\sum_{l=1}^{N_\kappa}\hat\phi^l_{\hat\kappa}(\hat{\mathbf{x}})\mathbf{U}^l_\kappa\right)\cdot DF_\kappa^{-1}\nabla\hat\phi^i_\kappa\,|\det DF_\kappa|\ \mathrm{d}\hat{\mathbf{x}}\ .\end{aligned}$$

An approximated quadrature formula is used for computing this integral. We denote by $\hat x_\alpha$ the quadrature points and by ω_α the weights.

$$\int_\kappa \mathbf{f}\left(\sum_{l=1}^{N_\kappa}\phi^l_\kappa(\mathbf{x})\mathbf{U}^l_\kappa\right)\cdot\nabla\phi^i_\kappa dV \approx \sum_\alpha\omega_\alpha\mathbf{f}\left(\sum_{l=1}^{N_\kappa}\hat\phi^l_{\hat\kappa}(\hat{\mathbf{x}}_\alpha)\mathbf{U}^l_\kappa\right)\cdot DF_\kappa^{-1}(\hat{\mathbf{x}}_\alpha)\nabla\hat\phi^i_\kappa(\hat{\mathbf{x}}_\alpha)\,|\det DF_\kappa(\hat{\mathbf{x}}_\alpha)|$$

1.4 Remarks on quadrature formulas

For linear problems, quadrature formulas can be chosen for being exact. For nonlinear problems, [**?**] suggests, for an approximation of degree p, to take a $(2p)$th order formula for cells, and a $(2p+1)$th order for faces.

For hypercube shapes, the optimal set of points (i.e. the one ensuring the highest degree with a given number of points) is the Gauss set of points. For other shapes, the optimal set of points is often unknown. A systematic way of deriving quadrature

formula on simplexes consists in using the image of Gauss points by Dubiner's map [**?**]. As far as we know, this is what is done in the quadrature formula proposed in [**?**]. However, the quadrature formulas obtained are not symmetric, and the number of points might be greater than other sets obtained by optimization procedures, see e.g. [**?**], and [**?**, **?**] for a list of quadrature formulas on simplexes.

2 The AGHORA and AEROSOL libraries

The AGHORA and AEROSOL libraries are two high order finite elements libraries that are respectively developed within ONERA and INRIA Bordeaux Sud Ouest (BACCHUS and CAGIRE teams). In this section, we give some details on these libraries.

2.1 The AGHORA library

The development of the AGHORA library began in 2012 with the PRF[1] of the same name. It is a high order (arbitrary order) finite elements library based on discontinuous elements. The models solved are the 3d compressible Euler or Navier-Stokes and RANS equations (with BR2 formulation for diffusive fluxes, see [**?**]) based on straight-sided and curved tetrahedra, hexahedra and prisms. It is developed in FORTRAN 95, and its development is based on previous experiences on two dimensional codes, that were developed for example within the ADIGMA project.

2.2 The AEROSOL library

The development of the AEROSOL library began at the end of 2010 with the PhD of Damien Genet. It became a shared project between the teams BACCHUS and CAGIRE in 2011. It is a library that aims at using tools that are developed mainly within Inria teams working on high performance computing at the Bordeaux center, see Figure 2.

2.2.1 Under the bonnet

AEROSOL is a high order finite elements library based on both continuous and discontinuous elements on hybrid meshes involving triangles and quadrangles in two dimensions and tetrahedra, hexaedra and prisms in three dimensions. More precisely, the finite element classes are generated until the fourth order polynomial approximation. Currently, it is possible to solve simple problems with the continuous Galerkin method (Laplace equation, SUPG stabilized advection equations), with the discontinous Galerkin method (Hyperbolic systems of first order) and with residual distribution schemes (scalar hyperbolic equations). It is written in C++, and its development started nearly from scratch as far as the general structure of the code is concerned. It strongly depends on the PAMPA library (see next section for details). It is linked with external linear solvers (up to now, PETSC[2] and MUMPS[3]). It is about to use the STARPU[4] task scheduler also being developed at INRIA. The choice of C++ allows for a good flexibility in terms of models and equations of state. Currently, the following models

[1] Projet de Recherche Fédérateur
[2] www.mcs.anl.gov/petsc/
[3] graal.ens-lyon.fr/MUMPS/
[4] starpu.gforge.inria.fr/

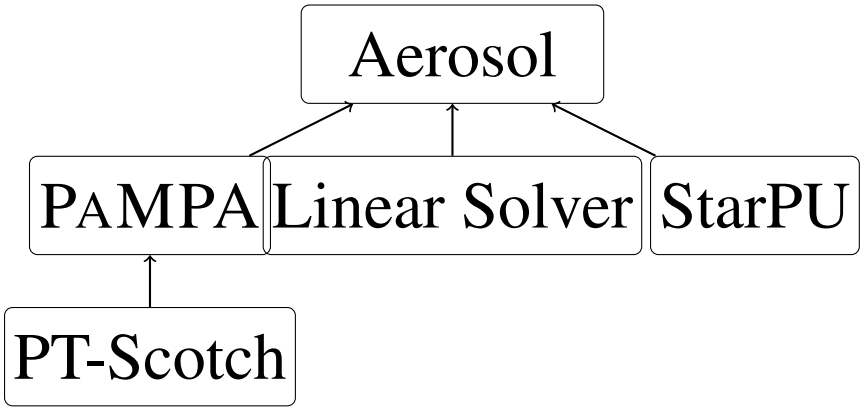


Figure 2: Structure of the AEROSOL code. It uses the PAMPA library for memory handling, for mesh partitioning, and for abstracting the MPI layer. It uses external linear solver (currently: PETSC and MUMPS). It is about to be linked with the STARPU library, which is a task scheduling library for hybrid architectures.

can be used: scalar advection, waves in a first order formulation, nonlinear scalar hyperbolic equation and Euler model with an abstract equation of state (currently: perfect gas and stiffened gas equation of state). It is working on linear elements, but the level of abstraction is sufficient for taking into account curved elements.

2.2.2 The PAMPA library

The PAMPA middleware library aims at abstracting mesh handling operations on distributed memory environments. It relieves solver writers from the tedious and error prone task of writing service routines for mesh handling, data communication and exchange, remeshing, and data redistribution. It is based on a distributed graph data structure that represents meshes as a set of *entities* (elements, faces, edges, nodes, etc.), linked by *relations* (that is, computation dependencies).

Given a numerical method based on a mesh, the user shall define an entity graph containing all the entities holding an unknown. For example, in a discontinuous Galerkin formulation, all the unknowns are located on the cells of the mesh, whereas in high order continuous finite elements, the unknowns might lie not only on elements, but also on points, edges and faces. Given the entity graph, PAMPA is able to compute a balanced partition of the unknowns (see Figure 4) and to compute adequate overlaps for data exchange across processors (see Figure 5). PAMPA handles mesh memory allocation, so that it can release memory after remeshing and/or mesh redistribution, as well as communication (roughly speaking, the user has almost no MPI to write in his code). PAMPA provides methods for iterating on mesh entities (e.g. on all local or boundary elements, on faces of an element, etc.), which eases the writing of numerical schemes based on finite elements methods. Last, PAMPA is also able to perform mesh adaptation in parallel, provided a sequential mesh adaptation software is linked to it; see Figure 3 for more details.

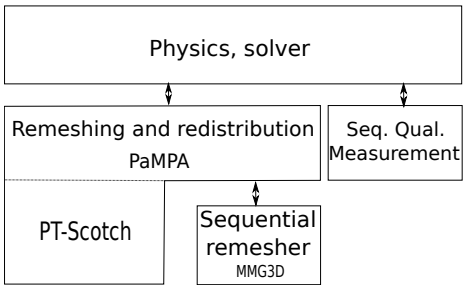


Figure 3: Overview of the use of PAMPA and of its interaction with other software. Solver writers (top box) rely on PAMPA for all mesh structure and data handling. PAMPA strongly depends on PT-SCOTCH for partitioning and redistribution. In order to perform (optional) dynamic mesh refinement with PAMPA, solver writers have to provide a sequential mesh refinement module (bottom box) that handles their types of elements, as well as a sequential quality measurement metric (rightmost box) to tell PAMPA where to perform remeshing. Remeshing is performed in parallel, after which the refined mesh is redistributed by PAMPA so as to re-balance computation load.

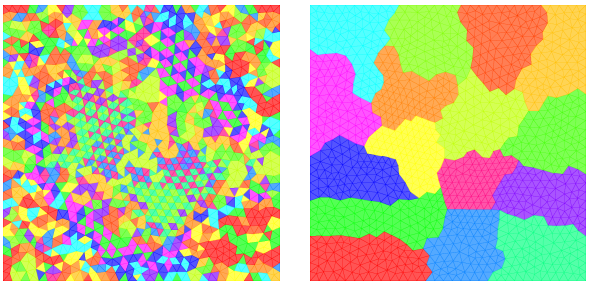


Figure 4: Example of mesh redistribution performed by PAMPA. Initially, each process reads a piece of a sequential mesh file (left picture), which may lead to poor data locality. Once the entity graph is given to PAMPA, it is able to compute a mesh partition that balances computation across processes and minimizes communication load (right picture).

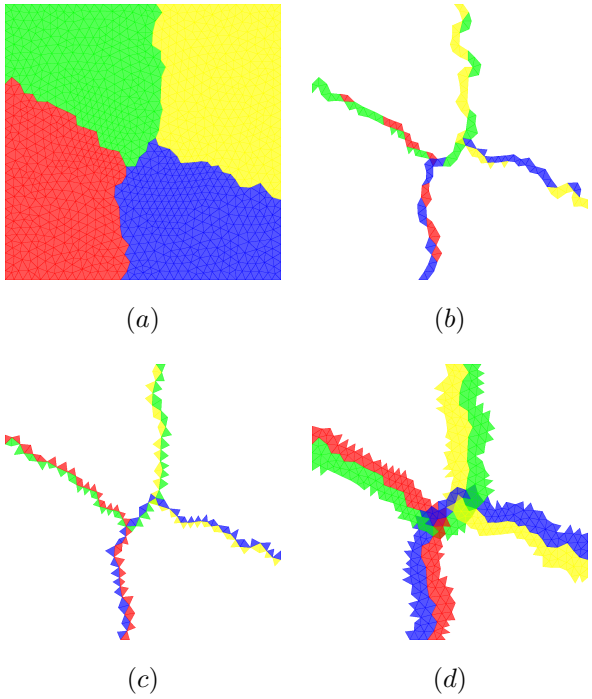


Figure 5: Examples of overlaps that can be computed by PAMPA according to the requirements of the numerical schemes. Fig. (a) represents the redistribution of the cells of a mesh across four processors, as yielded by PAMPA. Fig. (b) shows the overlap corresponding to a P^1 continuous finite element method (node-based neighbors), Fig. (c) shows the overlap for a discontinuous Galerkin or cell-centered finite volume method (face-based neighbors), and Fig. (d) shows the overlap for a high order finite volume scheme. Every processor of a given color stores its local data (Fig. (a)) plus the overlap cells of the same color.

2.3 Comparison of implementations

2.3.1 How to take into account the geometry

The main difference concerning implementations regards the strategy for taking into account geometries of cells in the different loops. For example, if we are concerned with the element loop, we have to compute

$$\int_\kappa \mathbf{f}\left(\sum_{l=1}^{N_\kappa}\phi_\kappa^l(\mathbf{x})\mathbf{U}_\kappa^l\right)\cdot\nabla\phi_\kappa^i dV \approx \sum_\alpha \omega_\alpha \mathbf{f}\left(\sum_{l=1}^{N_\kappa}\hat{\phi}_{\hat{\kappa}}^l(\hat{\mathbf{x}}_\alpha)\mathbf{U}_\kappa^l\right)\cdot DF_\kappa^{-1}\nabla\hat{\phi}_\kappa^i(\hat{\mathbf{x}}_\alpha)\,|\det DF_\kappa| \quad .$$

In this formula, some of the terms do not depend on the geometry of the cell: ω_α, $\hat{\phi}_{\hat{\kappa}}^l(\hat{\mathbf{x}}_\alpha)$, and $\nabla\hat{\phi}_\kappa^i(\hat{\mathbf{x}}_\alpha)$, whereas the following terms depend on the geometry: DF_κ^{-1} and $|\det DF_\kappa|$. The strategy in AGHORA was to store these geometrical dependent terms. This is very memory costly, because one needs to store it on all quadrature points (if elements are not linear, the geometrical terms are not constant in one given element), but it is often considered as paying off, as their evaluation is also costly. In the AEROSOL library, the only item stored is a pointer to the function that computes these geometrical terms. It is called for all cells at each computation step.

2.3.2 Computation of basis

In the same manner, the way to compute finite element basis is more generic in the AGHORA library. In the AEROSOL library, everything is based on the reference element. An orthogonal basis on the reference element is such that

$$\int_{\hat{\kappa}}\hat{\phi}_{\hat{\kappa}}^i(\hat{\mathbf{x}})\hat{\phi}_{\hat{\kappa}}^j(\hat{\mathbf{x}})\,\mathrm{d}\hat{\mathbf{x}} = \delta_{i,j}\int_{\hat{\kappa}}\left(\hat{\phi}_{\hat{\kappa}}^i(\hat{\mathbf{x}})\right)^2\,\mathrm{d}\mathbf{x} \quad ,$$

where $\delta_{i,j}$ is the Kronecker symbol. If we consider an element κ with associated reference elements $\hat{\kappa}$, i.e. such that $F_\kappa(\hat{\kappa}) = \kappa$, and if we consider the basis $\hat{\phi}_\kappa^i \circ F_\kappa^{-1}$, then

$$\begin{aligned}\int_\kappa \phi_\kappa^i(\mathbf{x})\phi_\kappa^j(\mathbf{x})\,\mathrm{d}\mathbf{x} &= \int_{\hat{\kappa}}\phi_\kappa^i\circ F_\kappa(\hat{\mathbf{x}})\phi_\kappa^j\circ F_\kappa(\hat{\mathbf{x}})\,|\det F_\kappa|\;\mathrm{d}\hat{\mathbf{x}}\\ &= \int_{\hat{\kappa}}\hat{\phi}_{\hat{\kappa}}^j(\hat{\mathbf{x}})\,|\det F_\kappa|\;\mathrm{d}\hat{\mathbf{x}} \quad .\end{aligned}$$

If F_κ is linear, then $|\det F_\kappa|$ is constant, and the basis $\hat{\phi}_\kappa^i \circ F_\kappa^{-1}$ is orthogonal on κ. But if F_κ is nonlinear, for example if κ is a hexaedra which is not a parallelepiped, or if κ is a curved simplex, then the basis $\hat{\phi}_\kappa^i \circ F_\kappa^{-1}$ may not be orthogonal. The library AGHORA is designed such as the finite element basis is always orthogonal.

For the sake of clarity, we introduce the construction of the orthonormal basis in the 1D case. The generalization to multi-space dimensions is straightforward. We start with a Taylor expansion of the numerical solution about x_κ the barycentre of the element κ:

$$\mathbf{u}_h(x,t) = \sum_{l=1}^{N_\kappa}\hat{\phi}_\kappa^l(x)\hat{\mathbf{U}}_\kappa^l(t), \quad \forall\, x\in\kappa,\, \forall t\geq 0 \quad , \tag{8}$$

where

$$\hat{\phi}^1_\kappa(x) = 1 \ , \quad \hat{\phi}^l_\kappa(x) = \frac{(x-x_\kappa)^{l-1} - \langle (x-x_\kappa)^{l-1} \rangle_\kappa}{(l-1)!} \ , \quad \forall 2 \leq l \leq N_\kappa \ , \quad x \in \kappa \ . \tag{9}$$

Functions $\hat{\phi}^l_\kappa$, for $2 \leq l \leq N_\kappa$ are centered monomials with zero mean value and $\langle \cdot \rangle_\kappa$ denotes the average operator over the element. The degrees of freedom in (8) are defined by

$$\hat{\mathbf{U}}^1_\kappa(t) = \langle u_h \rangle_\kappa \ , \quad \hat{\mathbf{U}}^l_\kappa(t) = \frac{\partial^{l-1} \mathbf{u}_h}{\partial x^{l-1}}\Big|_{x=x_\kappa} \ , \quad \forall 2 \leq l \leq N_\kappa \ , \quad t \geq 0 \ . \tag{10}$$

Functions $(\hat{\phi}^l_\kappa)_{1 \leq l \leq N_\kappa}$ are not orthogonal with respect to the inner product defined over the element κ and lead to a non-diagonal and ill-conditioned mass matrix $\mathbf{M}_\kappa$ even for $p \geq 1$ if $d \geq 2$. Their inversion for the time integration therefore requires extra computational costs and convergence properties of the method deteriorate. For general meshes, it is however possible to construct an orthonormal basis $(\phi^l_\kappa)_{1 \leq l \leq N_\kappa}$ in an arbitrary element κ by applying a Gram-Schmidt orthonormalization to the initial basis $(\hat{\phi}^l_\kappa)_{1 \leq l \leq N_\kappa}$. The new basis satisfies the following orthonormality property

$$(\phi^k_\kappa, \phi^l_\kappa)_\kappa = \int_\kappa \phi^k_\kappa(\mathbf{x}) \phi^l_\kappa(\mathbf{x}) dV = |\kappa| \delta_{kl} \ , \quad \forall 1 \leq k, l \leq N_\kappa \ , \tag{11}$$

and the mass matrix reduces to a diagonal matrix of the form $\mathbf{M}_\kappa = |\kappa| \mathbf{I}$. We refer to [**?**, **?**] for details on this procedure. As a consequence of the orthonormalization algorithm, both basis in expansions (4) and (8) are related by the following relation

$$\hat{\phi}^l_\kappa(x) = \sum_{k=1}^{l} r^{lk}_\kappa \phi^k_\kappa(x) \ , \quad \forall 1 \leq l \leq N_\kappa \ , \quad \forall x \in \kappa \ , \tag{12}$$

where $r^{lk}_\kappa = (\hat{\phi}^l_\kappa, \phi^k_\kappa)_\kappa$, for $1 \leq k \leq l-1$, denotes the inner product (11), and $(r^{ll}_\kappa)^2 = (\hat{\phi}^l_\kappa, \hat{\phi}^l_\kappa)_\kappa - \sum_{k=1}^{l-1} (\hat{\phi}^l_\kappa, \phi^k_\kappa)^2_\kappa$.

2.3.3 Communication handling

AGHORA performs data exchange by means of point-to-point communications, whereas AEROSOL uses collective communications using the overlap data exchange routines provided by PAMPA. If POSIX THREADs are available, PAMPA can perform such collective communications asynchronously on a specific thread, thus overlapping communication with computation. We were not yet able to fully use this feature, as most high-speed MPI implementations do not support the `MPI_THREAD_MULTIPLE` model.

3 Comparisons

3.1 Numerical tests

Due to the fact that both the AEROSOL and AGHORA libraries are not yet mature, a simple test was chosen for comparing libraries. In AGHORA, only three dimensional computations are possible. One and two dimensional computations are also possible, but by extruding one layer of cells in the z direction (and also in the y direction in one

dimension). In AEROSOL, true one and two dimensional computations can be considered, but geometrical functions, finite elements functions, and mesh reading were not yet available for three dimensional shapes. For performing fair comparisons, the AGHORA point of view was adopted, and one layer hexaedral meshes were used for doing a two dimensional test. Thus, AEROSOL was extended to three dimensions.

3.2 Yee vortex

We consider the convection of an isentropic vortex in a 2D uniform and inviscid flow [**?**] with conditions $\rho_\infty = 1$, $\mathbf{u}_\infty = \mathbf{e_x}$ and $T_\infty = 1$. The domain is the unit square $\Omega = [0,1]^2$ with periodic boundary conditions. The initial condition consists in a perturbation of the uniform flow which reads in primitive variables

$$\begin{aligned}
\rho(\mathbf{x},0) &= \Big(1-(\gamma-1)\Big(\tfrac{M_\infty M_v r_c}{2}\exp(1-\tfrac{r^2}{r_c^2})\Big)^2\Big)^{\frac{1}{\gamma-1}} \ , \\
u(\mathbf{x},0) &= 1 - M_v y \exp(1-\tfrac{r^2}{r_c^2}) \ , \\
v(\mathbf{x},0) &= M_v x \exp(1-\tfrac{r^2}{r_c^2}) \ , \\
\mathrm{p}(\mathbf{x},0) &= \frac{1}{\gamma M_\infty^2}\rho(\mathbf{x},0)^\gamma \ ,
\end{aligned}$$

where $r^2 = (x-x_c)^2 + (y-y_c)^2$ denotes the distance to the vortex centre $(x_c, y_c)^\top$, r_c and M_v are the radius and strength of the vortex, and M_∞ is the Mach number of the freestream flow. The exact solution of this problem is a pure convection of the vortex at velocity $\mathbf{u}_\infty$.

Numerical results are obtained for physical parameters $M_\infty = 0.5$, $M_v = 4$, $r_c = 0.1$, and a final time $T = 1$.

As the periodic boundary conditions were not yet available in the AEROSOL library, the test was slightly modified as follows: instead of adding a mean flow to the vortex, the initial state is taken as the state at infinity, without any velocity. On the boundaries, the values of the vortex are weakly applied. From a computational point of view, the work load is nearly the same, except for the communications that are needed for periodic boundary conditions in the case of the unsteady vortex.

Table 1 presents a weak scalability analysis where we evaluate the elapsed time observed for the global computation and for a fixed number of 24×24 hexaedral elements per computing core. Results are shown as a function of the number of cores. The relative increase, E, of the elapsed time with respect to the single core is also indicated shown, on Figure 6. Note that the memory requirement increases with the polynomial degree: the number of degrees of freedom per core is 4608, 15552, and 36864 for $p = 1$, 2, and 3, respectively. We observe that the solver scales nearly perfectly for $p = 4$, while results deteriorate for lower polynomial degrees $p = 1$ and 2. The relative part of communications compared to the numerical scheme decreases as the polynomial degree increases : from 1.1% for p =1 to 0.1% for p=4, as we can see in Table 2. As a consequence, the high frequency of communications between processes for a large number of cores becomes insensitive for large p-values. AEROSOL has a stronger work load for boundary sides, because in AGHORA, the boundary sides of the top and the bottom are ignored in the boundary side loop, whereas in AEROSOL, a freestream boundary condition was used.

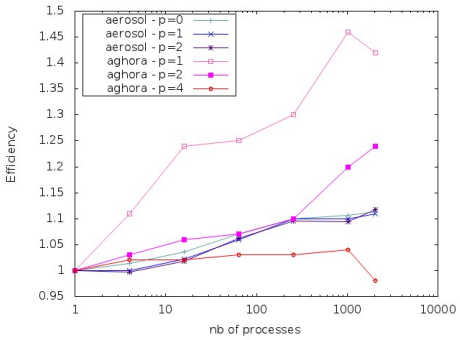


Figure 6: Comparison of the weak scalability obtained for AEROSOL and AGHORA with the library MVAPICH2. All the results show good scalability behavior. The least efficient is AGHORA for $p = 1$, and the most efficient is AGHORA for $p = 4$. Nevertheless, if we consider the results given in Table 1, we see that AEROSOL is much more costly if execution time is considered. This explains that the weak scalability is better, as AEROSOL has more computations for hiding communications.

We observe in Table 1 that AGHORA has lower execution time for a same number of RK substeps. Using an analyzer, we found the following reasons for the lower efficiency of AEROSOL

- First, AEROSOL is a finite element code that can both use continuous and discontinuous finite elements. In continuous finite elements, a degree of freedom is shared by many elements, if it is located on edges, faces, or points. Consequently, a connectivity table must be built for knowing which degrees of freedom is shared by which element. In AEROSOL, this connectivity table is never stocked, and its computation is general for continuous and discontinuous finite elements (thus it was not optimized for discontinuous finite elements). We found that the computation of this connectivity table is nearly three times more costly that the computation of the local residual. We are currently developing a version in which the connectivity table is stocked.

- Second, in AEROSOL, the geometry (e.g. local Jacobian of elements) is never stocked. This means that at each time step, and on all quadrature points, all the needed geometry is computed. In the computation of the local residual, we evaluated that 35% of the time elapsed in the computation of the local residual is for computing geometrical functions. We will develop in the near future a version in which the geometry is stocked.

- Last, we did not use any optimization option for compiling AEROSOL.

Table 2 gives a detailed analysis of the relative costs of different stages of the numerical algorithm. Results are given for one core. The implementation of surface and volume integrals for fluxes clearly represents the most expensive stage and its relative cost increases with p. This result is in agreement with the high scalability observed for high p-values as local operations strongly dominate communications.

Last, we want to compare the results obtained with different MPI libraries; however, AEROSOL needs an MPI library for which the `MPI_THREAD_MULTIPLE` is supported (i.e. if the process is multithreaded, multiple threads may call MPI at once with no restrictions), and such a functionality is only available with MVAPICH2 on

Table 1: Yee vortex problem: time/proc. [s] with Aerosol and Aghora for 3000 Runge-Kutta substeps.

	# cores	1	4	16	64	256	1024	2048
$p = 0$	Aerosol	202.52	205.32	209.91	216.68	222.69	224.01	225.52
$p = 1$	Aghora	12.67	14.02	15.69	15.88	16.42	18.43	17.97
	Aerosol	977.3	977.1	999.3	1035.6	1074.5	1074.3	1083.9
$p = 2$	Aghora	94.12	96.90	100.08	100.72	103.28	112.91	116.75
	Aerosol	5938.6	5923.2	6047.1	6307.7	6504.6	6501.0	6640.5
$p = 4$	Aghora	1401.5	1421.3	1430.2	1435.5	1439.1	1451.4	1370.8

Table 2: Yee vortex problem: relative costs in percent of different stages of the numerical algorithm per physical time step.

p	0	1		2		4
	Aerosol	Aerosol	Aghora	Aerosol	Aghora	Aghora
boundary surface integral	31.1	25.0	4.2	16.5	3.2	2.2
internal surface integral	51.3	36.4	57.5	21.2	46.8	34.6
volume integral	17.3	37.6	36.5	61.4	49.3	63.0
mass matrix inversion	0.3	1.0	1.8	0.8	0.7	0.2

	# cores	1	4	16	64	256	1024	2048
IntelMPI	time/proc. [s]	282.78	288.55	299.80	303.38	309.13	343.27	427.47
	E [-]	1	1.02	1.06	1.07	1.09	1.21	1.51
OpenMPI	time/proc. [s]	282.92	291.50	299.53	302.27	309.67	334.44	341.93
	E [-]	1	1.03	1.06	1.07	1.10	1.18	1.21
MVAPICH2	time/proc. [s]	282.37	290.71	300.25	302.17	309.85	338.73	350.26
	E [-]	1	1.03	1.06	1.07	1.10	1.20	1.24

Table 3: Comparison of execution time by core for a polynomial degree equal to 2 (9000 RK substeps). We observe that the most efficient execution is obtained for the OpenMPI library on AVAKAS.

# core	256	1024	2048
IntelMPI	15	122	5001
OpenMPI	05	010	0020
MVAPICH2	33	122	0536

Table 4: Average elapsed time (s) required for initialisation of MPI universe.

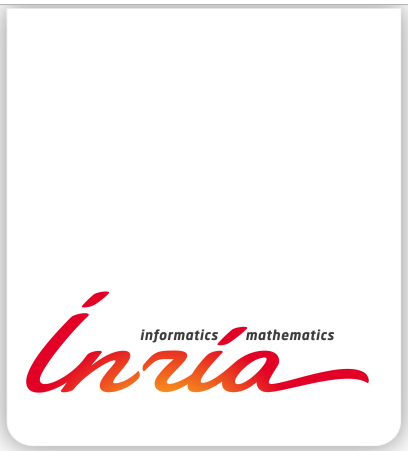
AVAKAS. That is why we show the results of this comparison only for AGHORA and not for AEROSOL, see Table 3. Note that an important part of this execution time can be due to the initilization of the MPI universe, see Table 4.

Conclusion

This project was the opportunity for us to compare the efficiency of the libraries AEROSOL and AGHORA. A weak scalability analysis was performed for comparing the two codes. We found that the building of the local connectivity and the geometry was very costly, and can explain a gap between the performances of the two codes. Concerning the weak scalability, we found that both of the codes have a good behavior, at least until 2048 cores.

Further than the results presented here, this project allowed us to have discussions on the way to develop and to factorize the code, that are difficult to account for here.

Acknowledgements Part of the computer time for this study was provided by the computing facilities MCIA (**Mésocentre de Calcul Intensif Aquitain**, on the cluster AVAKAS) of the Université de Bordeaux and of the Université de Pau et des Pays de l'Adour. Part of the computations have been performed at the **Mésocentre d'Aix-Marseille Université**.



RESEARCH CENTRE
BORDEAUX – SUD-OUEST

351, Cours de la Libération
Bâtiment A 29
33405 Talence Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399