ESAIM: PROCEEDINGS, December 2013, Vol. 43, p. 1-16
S. Descombes, B. Dussoubs, S. Faure, L. Gouarin, V. Louvet, M. Massot, V. Miele, Editors

COMPARISON OF HIGH ORDER ALGORITHMS IN AEROSOL AND AGHORA FOR COMPRESSIBLE FLOWS *

D. A. Mbengoue[1], D. Genet[1], C. Lachat[1], E. Martin[2], M. Mogé[1,3], V. Perrier[1,3], F. Renac[2], F.s Rué[1] et M. Ricchiuto[1]

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.

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 element methods for compressible flows that have been developed at ONERA and at INRIA for about one year, within the Aghora and Aerosol libraries.

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 volume 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. Indeed, RANS model 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 for low Reynolds numbers only (since the number of cells increases as $\mathrm{Re}^{9/4}$) and/or for academic configurations, for which efficient finite difference 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 performing 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.

* *Corresponding author : Vincent Perrier, e-mail :* vincent.perrier@inria.fr

[1] Inria Bordeaux Sud Ouest, 200 Avenue de la Vieille Tour, 33 405 Talence Cedex

[2] ONERA, The French Aerospace Lab, 92 320 Châtillon Cedex, France

[3] LMAP UMR 5142 CNRS, Université de Pau et des Pays de l'Adour, 64 013 Pau Cedex

Article published online by EDP Sciences and available at http://www.esaim-proc.org or http://dx.doi.org/10.1051/proc/201343001

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. The following methods can be considered.

- **High order finite volume methods**
  In finite volume approximations, the solution is considered as piecewise constant. For increasing the order of the scheme, a higher 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 take into account more neighboring cells. Moreover, when dealing with a hyperbolic problem, a *non oscillatory* interpolation [15] is required, 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 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, with the SUPG method [9] for example. Nevertheless, this method depends on parameters that can be difficult to tune.
- **Residual Distribution schemes**
  The development of fluctuation splitting/residual distribution schemes began with the early work of Phil Roe [19]. 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 [13]. 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 [4]. Although this approach has shown great potential in steady applications [2, 3], the current state of the art [1] shows that further work is needed to bring the method to the level of maturity of more popular techniques, such as the discontinuous Galerkin (DG) method.
- **Discontinuous Galerkin method**
  The development of discontinuous Galerkin method for nonlinear hyperbolic equations began in [10]. Its stabilization for flows with shocks was developed in the 90's, mainly by Cockburn and Shu (see [11] for a review). At the same time, a solution for dealing with Navier-Stokes equations was proposed in [7]. In spite of its cost (it involves much more degrees of freedom than classical continuous finite element methods), it is attractive because
  - it is naturally L^2 stable for linear problems,
  - a cell entropy inequality can be proven [16],
  - it has a compact stencil so that it has a good behavior in parallel environment [8].

  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 [17] for recent developments on this topic.

The library AGHORA [1] is a high order finite element library based on discontinuous discretizations and orthogonal basis developed at ONERA. AEROSOL [2] is a high order finite element library developed at INRIA. It can handle both continuous and discontinous discretizations. Until now, at once continuous Galerkin, discontinuous Galerkin and Residual Distribution schemes have been successfully implemented. In this article, we propose to compare the performances of the discontinuous Galerkin discretizations with orthogonal basis implemented in the libraries AGHORA and AEROSOL.

Organization of the article

The first section is dedicated to the description of the discontinuous Galerkin method for Euler equations. In the second section, we describe the structure and parallelization strategies of the libraries AEROSOL, developed at Inria, and AGHORA, developed at ONERA. In the third section, the performances of the libraries are compared in term of computational cost on one process and weak scalability.

1. Discontinuous Galerkin methods

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 of the problem (1) is projected on a space of piecewise continuous polynomials. 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

[1] Algorithm Generation for High Order Resolution in Aerodynamics

[2] AEROnautical SOLver

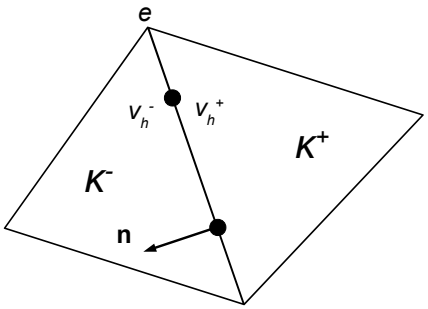

FIGURE 1. Inner and exterior elements κ^+ and κ^- and definition of traces $v_h^\pm$ on the interface e and of the unit outward normal vector $\mathbf{n}$.

$\hat{\kappa}$ through the mapping F_κ: simplex (line, triangle or tetrahedron), tensor elements (quadrangle, hexahedron), 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,
- a tensor product of one dimensional basis in the case of tensor elements,
- a tensor product of one dimensional basis and two dimensional basis on a triangle if $\hat{\kappa}$ is a prism,
- 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:

$$\text{find } \mathbf{u}_h \text{ in } [\mathcal{V}_h^p]^{d+2} \text{ such that for all } v_h \text{ in } \mathcal{V}_h^p \text{ we have } \int_{\Omega_h} v_h \partial_t \mathbf{u}_h d\mathbf{x} + \mathcal{B}_h(\mathbf{u}_h, v_h) = 0 \ . \tag{5}$$

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

$$\begin{aligned} \mathcal{B}_h(\mathbf{u}_h, v_h) \quad = \quad & -\int_{\Omega_h} \mathbf{f}(\mathbf{u}_h)\cdot\nabla v_h dV \\ & + \int_{\mathcal{E}_i} [\![v_h]\!] \tilde{\mathbf{f}}(\mathbf{u}_h^+, \mathbf{u}_h^-, \mathbf{n}) dS \\ & + \int_{\mathcal{E}_b} v_h^+ \mathbf{f}(\mathbf{u}_b(\mathbf{u}_h^+, \mathbf{n}))\cdot\mathbf{n} dS \ . \end{aligned} \tag{6}$$

In (6), $\mathbf{n}$ denotes the unit outward normal vector to an element κ^+ (see Figure 1) and $\mathbf{u}_b$ is an appropriate operator which allows imposing 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 [11] for instance). Still in (6), we 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 time discretization in equation (5) can then be written as

$$\mathcal{M}_\kappa \left(\frac{\mathbf{u}_h^{n+1} - \mathbf{u}_h^n}{\delta t} \right) + \mathcal{B}_h(\mathbf{u}_h^n, v_h) = 0 \ . \tag{7}$$

for the explicit Euler time stepping, the extension to another explicit time stepping being straightforward. In (7) $\mathcal{M}_\kappa$ denotes the local mass matrix defined as

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

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 [20, 22].

1.3. **Algorithm**

As an explicit Runge-Kutta time stepping is chosen, the algorithm consists in computing the spatial residual, and 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 degrees of freedom of a given element are on the same process (which is always the case). We now focus on the different loops that must be made for computing the local spatial residual defined on (6). We detail the computations involved in the element 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 $\mathcal{P}^p(\hat{\kappa})$. Using the definition of the basis given in Section 1.2

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

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

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

An approximated quadrature formula is used for computing this integral. We denote by $\hat{\mathbf{x}}_\alpha$ the quadrature points and by ω_α the weights. This yields

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

1.4. **Remarks on quadrature formulas**

For linear problems, quadrature formulas can be chosen for being exact. For nonlinear problems, [11] suggests, for an approximation of degree p, taking a $(2p)$th order formula for cells, and a $(2p+1)$th order formula 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 set of Gauss 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 [14]. As far as we know, this is what is done in the quadrature formula proposed in [21]. However, the quadrature formulas obtained are not symmetric, and the number of points might be greater than other sets

obtained by optimization procedures. The reader is referred to [25], and [12,23] for a list of quadrature formulas on simplexes.

2. The Aghora and Aerosol libraries

The Aghora and Aerosol libraries are two high order finite element libraries that are developed within Onera and Inria Bordeaux Sud Ouest (Bacchus and Cagire teams) respectively. In this section, we give some details on these libraries.

2.1. **The Aghora library**

The development of the Aghora code began in 2012 with the PRF[3] of the same name. It is a high order (arbitrary order) finite element library based on discontinuous elements. The code solves the 3d compressible Euler and Navier-Stokes equations with different modelization levels such as DNS, LES and RANS approaches. The LES method uses either classical closure to model the small scale dynamics, or a variational multiscale approach where the scale partitioning is set *a prori* in the function space. Its development is based on previous experiences on two dimensional codes, that were developed within the Adigma project. In contrast to the Aerosol library, the Aghora code does not use external libraries, but its own implementations for the MPI layer, the linear solver, etc. It is mainly developed in Fortran 95, the array pointers declaration statement respecting Fortran 2003 standard.

The space discretization is based on straight-sided or curved tetrahedra, hexahedra, prisms and pyramids and uses an orthonormal basis with either analytical bases (Legendre and Jacobi polynomials), or a numerical basis obtained from a Gram-Schmidt orthonormalization. The diffusive fluxes are discretized with the second Bassi-Rebay formulation (see [6]). Explicit time integration is achieved by using strong stability preserving Runge-Kutta schemes, while implicit time integration is performed with a backward Euler scheme for steady-state solutions, or with implicit Runge-Kutta (ESDIRK) schemes for time-dependent solutions. An in-house Newton-Krylov method has been implemented as linear solver and parallelized. It is based on a restarted GMRES with ILU0 preconditioning and a Jacobian-free approach for the matrix-vector product. Blas and Lapack libraries are used for performing basic vector and matrix operations.

The parallel strategy is a Single Program Multiple Data (SPMD) execution model based on a domain decomposition method. The MPI Point-to-Point communications respect the synchronous non-blocking send mode (MPI_Issend / MPI_Irecv) to overlap communications with computations. A specific data structure is devoted to the management of the MPI buffers for each domain, and spares some extra data copies to buffers before the send operation. The MPI Collective communications are reduced to the minimum to prevent a possible loss of efficiency at large scale. Several MPI distributions have already been experimented such as SGI MPT, IntelMPI, OpenMPI or mvapich2. In 2013, we plan to implement an hybrid MPI/OpenMP parallelization starting from a fine-grained classical approach to a coarse-grained approach. We expect a reduction of the footprint memory and the memory consumption, but also a better scalability at large scale as the number of involved MPI processes will be divided by the number of threads.

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.

[3]Projet de Recherche Fédérateur

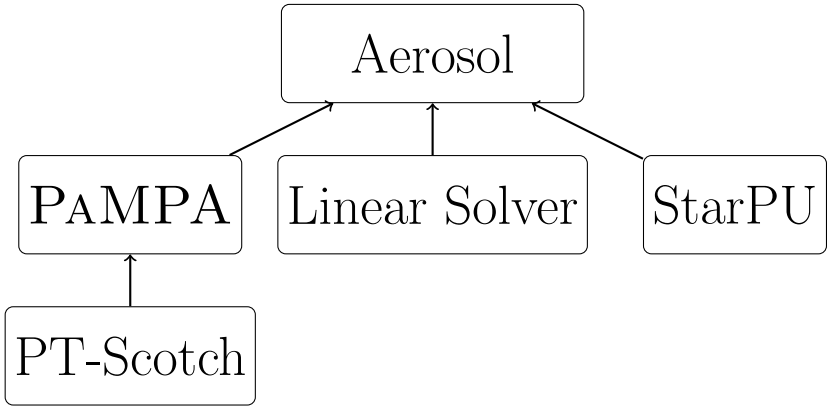

FIGURE 2. Structure of the AEROSOL code.

2.2.1. *Under the bonnet*

AEROSOL is a high order finite element library based on both continuous and discontinuous elements on hybrid meshes involving triangles and quadrangles in two dimensions and tetrahedra, hexahedra and prisms in three dimensions. More precisely, the finite element classes are generated up to 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 (first order hyperbolic systems) 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 for memory handling, for mesh partitioning, and for abstracting the MPI layer (see next section for details). It is linked with external linear solvers (up to now, PETSC[4] and MUMPS[5]). It is about to use the STARPU[6] task scheduler for hybrid architectures. STARPU is 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 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 works on linear elements, but the level of abstraction is sufficient for taking into account curved elements.

In the next section, we focus on the features of the library PAMPA, because contrary to the other midlevel libraries used in AEROSOL, no reference paper exists yet.

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 element, 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

[4] www.mcs.anl.gov/petsc/
[5] graal.ens-lyon.fr/MUMPS/
[6] starpu.gforge.inria.fr/

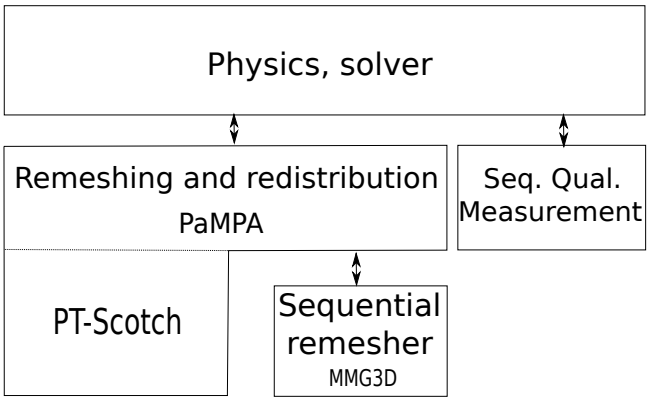

FIGURE 3. Overview of the use of PAMPA and of its interaction with other software.

the unknowns (see Figure 4) and to compute adequate overlaps for data exchange across processors (see Figure 5).

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

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 element 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: 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 of Figure 3) that handles their types of elements, as well as a sequential quality measurement metric (rightmost box of Figure 3) 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.

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

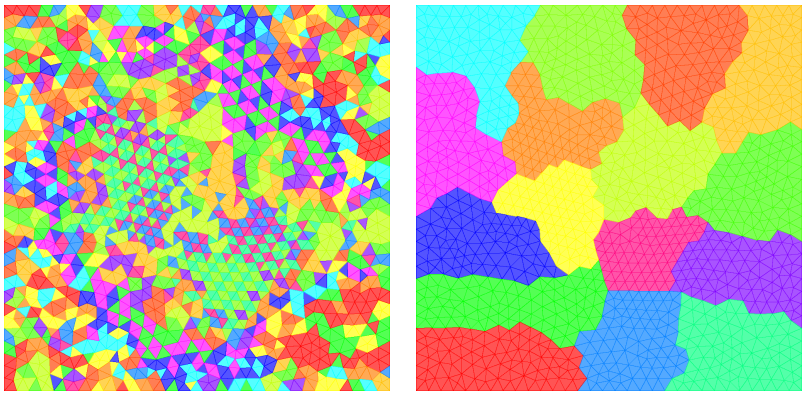

FIGURE 4. Example of mesh redistribution performed by PAMPA: random distribution (left picture) versus optimized distribution (right) for balancing computations and minimizing communications.

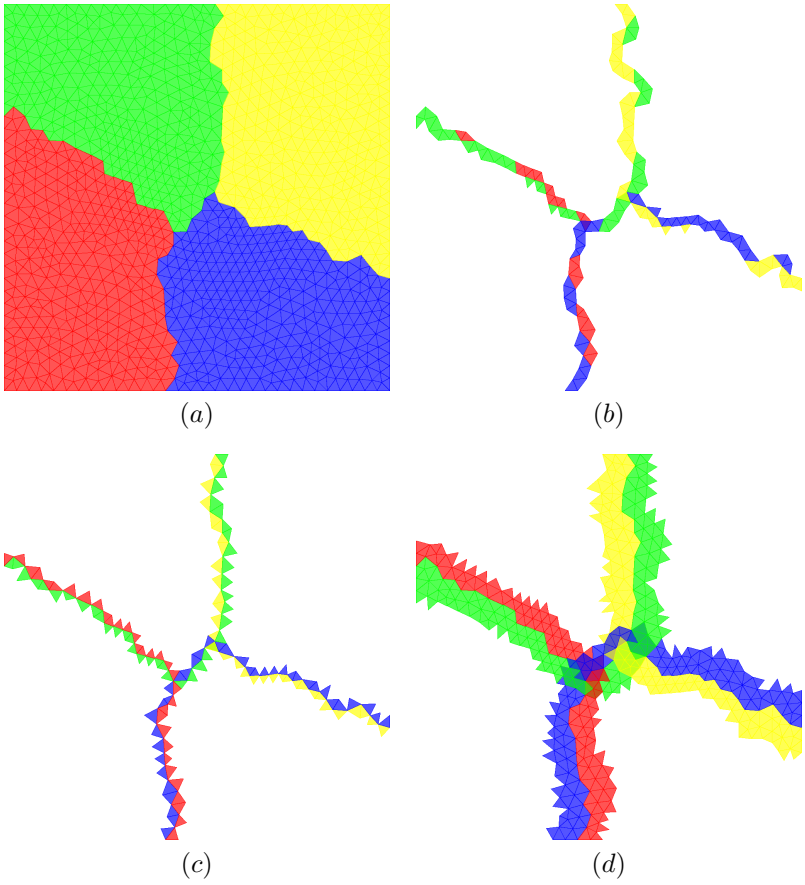

FIGURE 5. Examples of overlaps that can be computed by PAMPA according to the requirements of the numerical schemes.

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 this section, we focus on the way to define the finite element basis. An attractive property of the discontinuous Galerkin method is the structure of the mass matrix: it is block diagonal, with as many blocks as the number of elements. Each block is square and its size is the number of degrees of freedom of the element with which it matches. The aim of this section is to compare how the finite element basis are computed in Aerosol and Aghora.

In the Aerosol library, everything is based on the reference element. An orthogonal basis on the reference element is such that

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

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 $\hat{\phi}^i_\kappa \circ F_\kappa^{-1}$, then

$$\begin{aligned}\int_\kappa \phi^i_\kappa(\mathbf{x})\phi^j_\kappa(\mathbf{x})\,\mathrm{d}\mathbf{x} &= \int_{\hat{\kappa}} \phi^i_\kappa\circ F_\kappa(\hat{\mathbf{x}})\phi^j_\kappa\circ F_\kappa(\hat{\mathbf{x}})\,|\det F_\kappa|\,\mathrm{d}\hat{\mathbf{x}}\\ &= \int_{\hat{\kappa}} \hat{\phi}^i_{\hat{\kappa}}(\hat{\mathbf{x}})\hat{\phi}^j_{\hat{\kappa}}(\hat{\mathbf{x}})\,|\det F_\kappa|\,\mathrm{d}\hat{\mathbf{x}}\ .\end{aligned}$$

If F_κ is linear, then $|\det F_\kappa|$ is constant, and the basis $\hat{\phi}^i_\kappa \circ F_\kappa^{-1}$ is orthogonal on κ. But if F_κ is nonlinear, for example if κ is a hexahedra which is not a parallelepiped, or if κ is a curved simplex, then the basis $\hat{\phi}^i_\kappa \circ F_\kappa^{-1}$ may not be orthogonal.

The library Aghora is designed such as the finite element basis is always orthogonal. Here, we explain how to obtain such a basis. 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. The degrees of freedom of the unknown function $\mathbf{u}_h$ are defined as

$$\hat{\mathbf{U}}^1_\kappa(t) = \langle u_h\rangle_\kappa\ ,\quad \hat{\mathbf{U}}^l_\kappa(t) = \frac{\partial^{l-1}\mathbf{u}_h}{\partial x^{l-1}}\Big|_{x=x_\kappa}\ ,\quad \forall 2\le l\le N_\kappa\ ,\quad t\ge 0\ . \tag{8}$$

We start with a Taylor expansion of the numerical solution about x_κ the barycentre of the element κ:

$$\mathbf{u}_h(x,t) = \sum_{l=1}^{N_\kappa} \hat{\phi}^l_\kappa(x)\hat{\mathbf{U}}^l_\kappa(t),\quad \forall\, x\in\kappa,\,\forall t\ge 0\ , \tag{9}$$

where

$$\hat{\phi}^1_\kappa(x) = 1\ ,\quad \hat{\phi}^l_\kappa(x) = \frac{(x-x_\kappa)^{l-1} - \langle (x-x_\kappa)^{l-1}\rangle_\kappa}{(l-1)!}\ ,\quad \forall 2\le l\le N_\kappa\ ,\quad x\in\kappa\ . \tag{10}$$

Functions $\hat{\phi}^l_\kappa$, for $2\le l\le N_\kappa$ are centered monomials with zero mean value and $\langle\cdot\rangle_\kappa$ denotes the average operator over the element.

Functions $(\hat{\phi}^l_\kappa)_{1\le l\le N_\kappa}$ 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 $p\ge 1$ if $d\ge 2$. 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 $(\phi^l_\kappa)_{1\le l\le N_\kappa}$ in an arbitrary element κ by applying a Gram-Schmidt orthonormalization to the initial basis $(\hat{\phi}^l_\kappa)_{1\le l\le N_\kappa}$. The new basis satisfies the following orthonormality property

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

and the mass matrix reduces to a diagonal matrix of the form $\mathbf{M}_\kappa = |\kappa|\mathbf{I}$. We refer to [5, 18] for details on this procedure. As a consequence of the orthonormalization algorithm, both basis in expansions (4) and (9) are related by the following relation

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

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

To conclude this section, on general meshes (namely possibly curved and possibly non parallepipedic elements), orthogonal basis cannot be obtained by simple transformation of an orthogonal basis on the reference element, because the transformation between one element and its associated reference element is not linear. Nevertheless, an orthogonal basis can be obtained by the Gram-Schmidt algorithm. The resulting basis strongly depends on the geometry of each element so that the values of the finite element basis on the quadrature points must be stocked *for all elements*, and not only for the reference elements.

2.3.3. *Communication handling*

Aghora performs data exchange by means of point-to-point communications (synchronous non-blocking send mode with MPI_Issend and MPI_Irecv), 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 in Aerosol, as most high-speed MPI implementations do not support the `MPI_THREAD_MULTIPLE` model.

3. Comparisons

The aim of this section is to assess the impact of the different strategies described in the previous section on the execution time.

3.1. **Numerical test: Yee vortex**

In this section, we present a simple test case for comparing performances of both libraires. We consider the convection of an isentropic vortex in a 2D uniform and inviscid flow [24] with conditions $\rho_\infty = 1$, $\mathbf{u}_\infty = \mathbf{e}_\mathbf{x}$ 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 $T = 1$.

3.2. **Implementation of the test in the libraries**

The test described in the previous section is two dimensional. In Aghora, only three dimensional meshes can be handled. One and two dimensional computations must be done 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 element functions, and mesh reading were not yet available for three dimensional shapes. Nevertheless, in a high order framework, computations on a two dimensional mesh are much less costly than computations on a three dimensional mesh obtained by a one layer extrusion of a two dimensional mesh. Indeed for an approximation of degree p, and when dealing with a tensor product cell of dimension d (quadrangle for $d = 2$ and hexahedron for $d = 3$), the number of degrees of freedom is $(p + 1)^d$, the number of volume quadrature points is $(p + 1)^d$, and the number of face quadrature points is $(p+1)^{d-1}$. Note that this problem is really specific to high order methods: for first order finite volume methods, which are the discontinuous Galerkin methods with $p = 0$, computations on two dimensional meshes are nearly as costly as computations on three dimensional meshes obtained by a one layer extrusion of a two dimensional mesh.

As we are dealing here with high order methods, and in order to perform fair comparisons, the Aghora point of view was adopted, and one layer hexahedral meshes were used for doing this two dimensional test. Thus, Aerosol was extended to three dimensions.

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.

3.3. **Results**

Table 2 presents a weak scalability analysis where we evaluate the elapsed time observed for the global computation and for a fixed number of 24×24 hexahedral 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 on Figure 6. Note that the memory requirement increases with the polynomial degree: the number of degrees of freedom per core is 4608, 15552, 36864, 72000 for $p = 1$, 2, 3 and 4, respectively. We observe that the solver scales nearly perfectly for $p = 4$, while results deteriorate for lower polynomial degrees $p = 1$ and 2. 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.

3.3.1. *Analysis of the single core results*

If we focus on the single core results, we see that Aerosol is between two and three times more costly that Aghora. Actually the results found during the CEMRACS project were even worse, for two reasons:

(1) The library Aerosol is based on the high level library PaMPA. This library gives iterators that allows to loop on elements, faces, neighboring elements of faces, etc... In a first version of the code, PaMPA iterators were used for all the loops. The results on a single core, without optimization options, gave an execution time of 977.3 s for $p = 1$, and of 5938.6 s for $p = 2$. A new version was developed, where the connectivity is stored, and where PaMPA iterators are used only once for building the connectivity. Execution time was improved from 977.3 to 391.93 s for $p = 1$, and from 5938.6 s to 1450.59 s for $p = 2$.

(2) The first tests we did were done without activating compiler optimization options. These optimizations and some other classical tricks (removing some checkings, avoiding the call to complex C++ data structures, and using local registers) decreased the execution time from 391.93 s to 49.2 s for $p = 1$ and from 1450.59 s to 170.14 s for $p = 2$.

TABLE 1. Yee vortex problem: relative costs in percent of different stages of the numerical algorithm per physical time step on one process.

p	1		2		4	
	Aerosol	Aghora	Aerosol	Aghora	Aerosol	Aghora
boundary surface integral	28.9	4.2	20.3	3.2	9.5	2.2
internal surface integral	47.4	57.5	31.2	46.8	15.8	34.6
volume integral	23.1	36.5	48.1	49.3	74.5	63.0
mass matrix inversion	0.6	1.8	0.4	0.7	0.2	0.2

TABLE 2. Yee vortex problem: time/proc. [s] with Aerosol and Aghora for 3000 Runge-Kutta substeps with the MPI distribution MVAPICH2.

	# cores	1	4	16	64	256	1024	2048
$p=1$	Aghora	12.67	14.02	15.69	15.88	16.42	18.43	17.97
	Aerosol	49.20	50.76	57.97	69	76.97	77.8	
$p=2$	Aghora	94.12	96.90	100.08	100.72	103.28	112.91	116.75
	Aerosol	170.14	175.1	204.19	234.41	253.37	255.07	
$p=4$	Aghora	1401.5	1421.3	1430.2	1435.5	1439.1	1451.4	1370.8
	Aerosol	2079.3	2093.7	2134.8	2252.2	2315.3	2349.4	

As far as the remaining difference on a single core between AEROSOL and AGHORA is concerned (see first column of Table 2), we attribute this to the strategy of stocking all the geometry instead of recomputing it at each time step, as explained in Section 2.3.1.

3.3.2. *Analysis of the scalability*

Table 1 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.

Weak scalability results are shown on Figure 6. The worse efficiency is obtained for AEROSOL with $p = 1$, and the most efficient is AGHORA with $p = 4$. As explained in Section 2.3.3, communications are not implemented the same way in AEROSOL and in AGHORA. In AGHORA, point-to-point non blocking communications are used, whereas collective communications sent on PTHREAD are used in AEROSOL. It was not obvious to predict which communications would be the fastest:

- on one side, point-to-point communications send and receive the least data that are shared between two processes
- on the other side, collective communications send all the overlaps to all the processes. This means that processes receive some parts of the overlap they do not need. These collective communications are non-blocking because they are sent by a PTHREAD. Moreover, they might be more efficiently broadcast than with (MPI_Issend / MPI_Irecv) if the MPI distribution uses a tree broadcast algorithm.

Our conclusion for the test we did is that point-to-point communications are more efficient than collective communication sent on a PTHREAD.

3.3.3. *Influence of the MPI distribution used on the performances*

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

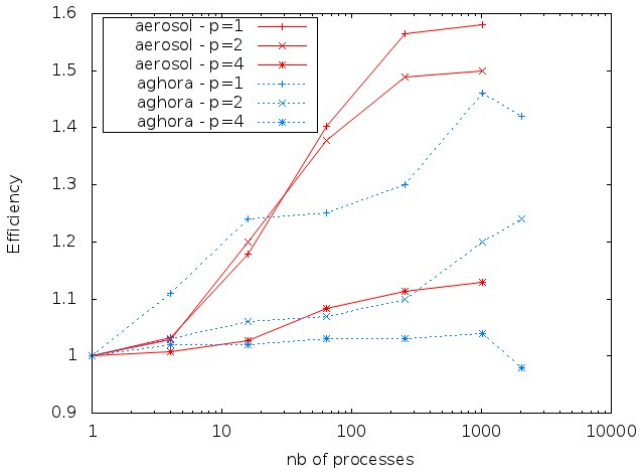

FIGURE 6. Comparison of the weak scalability obtained for AEROSOL (solid red line) and AGHORA (dashed blue line) with the library MVAPICH2.

TABLE 3. Comparison of execution time by core for a polynomial degree equal to 2 (9000 RK substeps) for the AGHORA library.

	# 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 4. Average elapsed time (s) required for initialisation of MPI universe on AVAKAS.

# core	256	1024	2048
INTELMPI	15	122	5001
OPENMPI	05	010	0020
MVAPICH2	33	122	0536

on AVAKAS. That is why we show the results of this comparison only for AGHORA and not for AEROSOL, see Table 3. We observe that the most efficient execution is obtained for the OPENMPI library on AVAKAS. 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

1024 cores. We compared two ways of doing non blocking communications: point-to-point communications, and collective communications sent on a PTHREAD. In our test, point-to-point communications were found to be more efficient.

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**.

REFERENCES

[1] R. Abgrall. A review of residual distribution schemes for hyperbolic and parabolic problems : the July 2010 state of the art. *Commun.Comput.Phys.*, 11(4):1043–1080, 2012.

[2] R. Abgrall, G. Baurin, P. Jacq, and M. Ricchiuto. Some examples of high order simulations of inviscid flows on unstructured hybrid meshes by residual distribution schemes. *Computers & Fluids*, 61:1–13, 2012.

[3] R. Abgrall, A. Larat, and M. Ricchiuto. Contruction of very high order residual distribution for steady inviscid flow problems on hybrid unstructured meshes. *J.Comput.Phys.*, 230(11):4103–4136, 2011.

[4] R. Abgrall and P.L. Roe. High-order fluctuation schemes on triangular meshes. *J. Sci. Comput.*, 19(3):3–36, 2003.

[5] F. Bassi, L. Botti, A. Colombo, D. A. Di Pietro, and P. Tesini. On the flexibility of agglomeration based physical space discontinuous Galerkin discretizations. *J. Comput. Phys.*, 231(1):45–65, 2012.

[6] F Bassi, A Crivellini, S Rebay, and M Savini. Discontinuous Galerkin solution of the Reynolds-averaged Navier-Stokes and $k-\omega$ turbulence model equations. *Computers & Fluids*, 34(4-5):507–540, MAY-JUN 2005. Workshop on Residual Distribution Schemes, Discontinuous Galerkin Schemes, Multidimensional Schemes and Mesh Adaptation, Univ Bordeaux I, Inst Math, Talence, FRANCE, JUN 23-25, 2002.

[7] F. Bassi and 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):267–279, 1997.

[8] R. Biswas, K. D. Devine, and J. Flaherty. Parallel, adaptive finite element methods for conservation laws. *Appl. Numer. Math.*, 14:255–283, 1994.

[9] Alexander N. Brooks and Thomas J. R. Hughes. Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations. *Comput. Methods Appl. Mech. Engrg.*, 32(1-3):199–259, 1982. FENOMECH '81, Part I (Stuttgart, 1981).

[10] Guy Chavent and Bernardo Cockburn. The local projection P^0P^1-discontinuous-Galerkin finite element method for scalar conservation laws. *RAIRO Modél. Math. Anal. Numér.*, 23(4):565–592, 1989.

[11] Bernardo Cockburn and Chi-Wang Shu. Runge-Kutta discontinuous Galerkin methods for convection-dominated problems. *J. Sci. Comput.*, 16(3):173–261, 2001.

[12] Ronald Cools. An encyclopaedia of cubature formulas. *J. Complexity*, 19(3):445–453, 2003. Numerical integration and its complexity (Oberwolfach, 2001).

[13] H. Deconinck and M. Ricchiuto. Residual distribution schemes: foundation and analysis. In E. Stein, R. de Borst, and T.J.R. Hughes, editors, *Encyclopedia of Computational Mechanics*. John Wiley & Sons, Ltd., 2007. DOI: 10.1002/0470091355.ecm054.

[14] Moshe Dubiner. Spectral methods on triangles and other domains. *J. Sci. Comput.*, 6(4):345–390, 1991.

[15] Ami Harten, Stanley Osher, Björn Engquist, and Sukumar R. Chakravarthy. Some results on uniformly high-order accurate essentially nonoscillatory schemes. *Appl. Numer. Math.*, 2(3-5):347–377, 1986.

[16] Guang Shan Jiang and Chi-Wang Shu. On a cell entropy inequality for discontinuous Galerkin methods. *Math. Comp.*, 62(206):531–538, 1994.

[17] Norbert Kroll, Heribert Bieler, Herman Deconinck, Vincent Couaillier, Harmen van der Ven, and Kaare Sørensen, editors. *ADIGMA - A European Initiative on the Development of Adaptive Higher-Order Variational Methods for Aerospace Applications*, volume 113 of *Notes on Numerical Fluid Mechanics and Multidisciplinary Design Volume*. Springer, 2010.

[18] Jean-François Remacle, Joseph E. Flaherty, and Mark S. Shephard. An adaptive discontinuous Galerkin technique with an orthogonal basis applied to compressible flow problems. *SIAM Rev.*, 45(1):53–72 (electronic), 2003.

[19] P.L. Roe. Fluctuations and signals - a framework for numerical evolution problems. In K.W. Morton and M.J. Baines, editors, *Numerical Methods for Fluids Dynamics*, pages 219–257. Academic Press, 1982.

[20] Chi-Wang Shu and Stanley Osher. Efficient implementation of essentially nonoscillatory shock-capturing schemes. *J. Comput. Phys.*, 77(2):439–471, 1988.

[21] Pavel Šolín, Karel Segeth, and Ivo Doležel. *Higher-order finite element methods*. Studies in Advanced Mathematics. Chapman & Hall/CRC, Boca Raton, FL, 2004. With 1 CD-ROM (Windows, Macintosh, UNIX and LINUX).

[22] Raymond J. Spiteri and Steven J. Ruuth. A new class of optimal high-order strong-stability-preserving time discretization methods. *SIAM J. Numer. Anal.*, 40(2):469–491 (electronic), 2002.
[23] A. H. Stroud. *Approximate calculation of multiple integrals*. Prentice-Hall Inc., Englewood Cliffs, N.J., 1971. Prentice-Hall Series in Automatic Computation.
[24] H. C. Yee, N. D. Sandham, and M. J. Djomehri. Low-dissipative high-order shock-capturing methods using characteristic-based filters. *J. Comput. Phys.*, 150(1):199–238, 1999.
[25] Linbo Zhang, Tao Cui, and Hui Liu. A set of symmetric quadrature rules on triangles and tetrahedra. *Journal of Computational Mathematics*, 27(1):89–96, JAN 2009.