ECCOMAS Congress 2016
VII European Congress on Computational Methods in Applied Sciences and Engineering
M. Papadrakakis, V. Papadopoulos, G. Stefanou, V. Plevris (eds.)
Crete Island, Greece, 5–10 June 2016

HYBRID OPENMP/MPI PARALLELIZATION OF A HIGH–ORDER DISCONTINUOUS GALERKIN CFD/CAA SOLVER

F. Bassi[1], A. Colombo[1], A. Crivellini[2] and M. Franciolini[2]

[1]Università degli Studi di Bergamo, Dipartimento di Ingegneria e Scienze Applicate
Viale Marconi 5, 24044, Dalmine (BG), Italy
e-mail: {francesco.bassi,alessandro.colombo}@unibg.it

[2] Università Politecnica delle Marche, Dipartimento di Ingegneria Industriale e Scienze Matematiche
Via Brecce Bianche 12, 60131, Ancona, Italy
e-mail: {a.crivellini@,m.franciolini@pm.}univpm.it

Keywords: Discontinuous Galerkin, parallelization, CFD, CAA, OpenMP, HPC.

Abstract. *This paper describes the implementation of an hybrid OpenMP/MPI parallelization strategy in a Discontinuous Galerkin solver used for DNS and LES or CAA computations, to fruitfully exploit the modern massively parallel HPC facilities. It is usually believed that the sheared memory view of OpenMP can easily increase the parallel efficiency of codes dealing with multi–core clusters. The idea consists of running calculations on those machines restricting as much as possible the use of the MPI library to the communications between nodes and exploiting the shared memory paradigm within a node. However, in practice, the achievement of a real parallel performance gain is not straightforward. Moreover, as far as DG solvers are concerned, almost nothing is reported in the current literature about the hybrid MPI/OpenMP implementation. In this work a colouring algorithm has been employed for OpenMP. The resulting hybrid strategy performs quite satisfactory, since generally it is more efficient of the pure MPI implementation. However, the performances are heavily dependent on hardware platforms, as well as on computational details such as the polynomial order of space discretization or the number of computational elements. Several scalability tests have been performed, resulting in the conclusion that the best performance can be achieved only with a proper choice of the number of MPI partition and OpenMP threads to be used within a single node. The reliability of the method was here assessed by solving the Taylor Green vortex problem at Reynolds numbers equal to 800 and 1600 and the Linear Euler acustic scattering from a rigid sphere.*

1 Introduction

In recent years Discontinuous Galerkin (DG) methods have emerged as one of the most promising high–order discretization techniques for CFD. DG methods have been successfully applied to the simulation of turbulent flows by solving the Reynolds averaged Navier–Stokes (RANS) equations [1, 2, 3], and more recently, have also been found very well suited for the Direct Numerical Simulation (DNS) as well as Implicit Large Eddy Simulation (ILES) of turbulent flows [4, 5] due to their favourable dispersion and dissipation properties. One of the advantages of DG methods is the use of a compact stencil, which is independent of the degree of polynomial approximation and is thus well suited for massively parallel implementations.

The most common strategy for exploiting HPC facilities relies on the usage of the Message Passing Interface (MPI) library, which is suitable for both the shared and distributed memory platforms. As an alternative, on shared memory systems only, the Open Multi-Processing (OpenMP) paradigm can be used. While the former strategy is widely used in the CFD/CAA community, almost nothing is reported in literature about the latter with reference to DG methods. To the authors' knowledge only [6] partially deals with this subject. However, even if OpenMP is confined to shared memory systems, some advantages of this parallelization strategy over the MPI approach are clear. The first one is the small programming effort required for the parallelization, at least partial, of an executable. In fact, it is possible to use few work–sharing directives in the most time consuming section of the source code. With this approach a relevant reduction of the execution time can be obtained almost immediately using the multi–core CPUs. At a later stage the developer can eventually improve the parallel performance considering additional pieces of source code, resulting in an incremental development. The second theoretical advantage refers to the expected performance gain achievable on multi– and many–core platforms.

Especially for modern HPC facilities, the use of the MPI library within a computing node can be a not optimal choice when compared to a sheared memory-oriented parallelism that can drastically reduce the cost of the inter-core communication. For this reason, a hybrid OpenMP/MPI implementation is considered to improve the parallel efficiency of codes running on clusters of multi–core nodes. However, several papers highlight that for a large number of cores the achievement of high OpenMP parallel efficiencies is not trivial and it is not clear if all the applications and hardware platforms can effectively benefit from the hybrid approach [7]. There are several reasons for those evidences: 1) the code should be fully parallelized even in the OpenMP context; 2) the use of OpenMP implies overheads due to the initialization of parallel and/or work–sharing regions; 3) the overheads due to the explicit and implicit synchronizations of some of the work-sharing constructs; 4) the penalization due to false sharing of cache lines between processors; 5) the reduction of memory bandwidth of cache coherent Non Uniform Memory Access (ccNUMA) hardware used for the more recent and common multi–core chips.

Some of these arguments shed some light on the great influence of the hardware characteristics on the parallel effectiveness. At this regards it is particularly relevant the number of NUMA regions within a node and the resulting latency encountered when transferring data from the random access memory (RAM) of one of these regions to the cache memory of the core owned by a different region. It is worth mentioning that, to date, OpenMP does not supply any instruction to force the placement of one data in a particular NUMA region. The use of a first touch policy within the source code or, alternatively, the use of some operating system utilities/commands, such as `numactl`, are viable shortcuts for reaching this objective. In fact, they are able only partially to control the affinity of a data to a socket. Even if they can improve the OpenMP performance, the effectiveness of these approaches may change with the platform hardware and its operating system. OpenMP efficiency is also strongly affected by the used compiler. The parallelization of the OpenMP directives is demanded to the compiler, thus depending on how efficiently it implements the standard. This is not the only compiler characteristic impacting the parallel efficiency. Just switching on the OpenMP option the effectiveness of the compiler optimizations may vary significantly mainly within a parallel loop. Additionally, the choiche of a proper parallelization strategy based on OpenMP is the key ingredient of an efficient solver. While with an MPI parallelization the use of a domain decomposition algorithm is almost unavoidable, the shared memory parallel programming context offers more opportunities. Here we explore a strategy commonly referred as colouring approach, which retains all the OpenMP advantages and it can be very simply introduced in a already existing MPI solver. Moreover, it is also well suited to deal with hardware accelerators, an opportunity considered in the current OpenMP 4.0 standard.

Despite DG discretization is particularly well suited for a pure MPI implementation, due to the compactness of the discretization which produces a limited amount of information exchanged between grid partitions, the proposed hybrid OpenMP/MPI implementation perform well as shown in the numerical tests reported in Section 6. However, the best performance can be achieved only with a proper choice of the number of partitions to be inserted within a single node. The advantages are mainly evident for large numbers of computational cores within a computing node, when the pure MPI implementation suffers for high communications overheads. Our results show at most a 38% gain in terms of parallel efficiency of the hybrid OpenMP/MPI implementation with respect to the pure MPI strategy.

2 DG approximation of the Navier–Stokes equations

The complete set of Navier–Stokes equations can be written as:

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_j}(\rho u_j) = 0, \tag{1}$$

$$\frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_j}(\rho u_j u_i + p\delta_{ji}) = \frac{\partial \tau_{ji}}{\partial x_j}, \tag{2}$$

$$\frac{\partial}{\partial t}(\rho e_0) + \frac{\partial}{\partial x_j}(\rho u_j h_0) = \frac{\partial}{\partial x_j}\left(u_i \tau_{ij} - q_j\right), \tag{3}$$

where the total energy e_0 and total enthalpy h_0, pressure p, the total stress tensor τ_{ij} and the heat flux vector q_j are given by

$$\begin{aligned}
e_0 &= e + u_k u_k/2, \\
h_0 &= e_0 + p/\rho, \\
p &= (\gamma - 1)\rho\left(e_0 - u_k u_k/2\right), \\
\tau_{ij} &= 2\mu\left(S_{ij} - \frac{1}{3}\frac{\partial u_k}{\partial x_k}\delta_{ij}\right), \\
q_j &= -\frac{\mu}{\mathrm{Pr}}\frac{\partial h}{\partial x_j}.
\end{aligned}$$

Here e is the internal energy, h is the enthalpy, $\gamma = c_p/c_v$ is the ratio of gas specific heats, Pr is the molecular Prandtl number and

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$$

is the mean strain-rate tensor.

The governing equations (1)–(3) can be written in the following compact form

$$\frac{\partial \mathbf{u}}{\partial t} + \boldsymbol{\nabla} \cdot \mathbf{F}_c(\mathbf{u}) + \boldsymbol{\nabla} \cdot \mathbf{F}_v(\mathbf{u}, \boldsymbol{\nabla}\mathbf{u}) = \mathbf{0}, \tag{4}$$

where $\mathbf{u}, \in \mathbb{R}^M$ denote the vectors of the M conservative variables and source terms, $\mathbf{F}_c, \mathbf{F}_v \in \mathbb{R}^M \otimes \mathbb{R}^N$ denote the inviscid and viscous flux functions, respectively, and N is the space dimension. In order to obtain the weak formulation of the equations, one can multiply equation (4) by an arbitrary test function ϕ and integrate over the domain Ω. Integrating by parts the divergence terms, the weak form of equation (4) reads:

$$\int_\Omega \phi \frac{\partial \mathbf{u}}{\partial t}\, \mathrm{d}\mathbf{x} - \int_\Omega \boldsymbol{\nabla}\phi \cdot \mathbf{F}(\mathbf{u}, \boldsymbol{\nabla}\mathbf{u})\, \mathrm{d}\mathbf{x} + \int_{\partial\Omega} \phi \mathbf{F}(\mathbf{u}, \boldsymbol{\nabla}\mathbf{u}) \cdot \mathbf{n}\, \mathrm{d}\sigma = \mathbf{0}, \tag{5}$$

where $\mathbf{F}$ is the sum of the inviscid and viscous fluxes.

In order to build a DG discretization of equation (5), we consider an approximation Ω_h of Ω and a triangulation $\mathcal{T}_h = \{K\}$ of Ω_h, *i.e.*, a set of ne non-overlapping elements K not necessarily simplices. We denote with $\mathcal{F}_h^i$ the set of internal element faces, with $\mathcal{F}_h^b$ the set of boundary element faces and with $\mathcal{F}_h = \mathcal{F}_h^i \cup \mathcal{F}_h^b$ their union. We moreover set

$$\Gamma_h^i = \bigcup_{F \in \mathcal{F}_h^i} F, \qquad \Gamma_h^b = \bigcup_{F \in \mathcal{F}_h^b} F, \qquad \Gamma_h = \Gamma_h^i \cup \Gamma_h^b. \tag{6}$$

The solution is approximated on $\mathcal{T}_h$ as a piecewise polynomial function possibly discontinuous on element interfaces, *i.e.*, we assume the following space setting for each component $u_{h_i} = u_{h_1}, \ldots, u_{h_M}$ of the numerical solution $\mathbf{u}_h$:

$$u_{h_i} \in \Phi_h \stackrel{\text{def}}{=} \left\{\phi_h \in L^2(\Omega) : \phi_h|_K \in \mathbb{P}_k(K)\ \forall K \in \mathcal{T}_h\right\} \tag{7}$$

for some polynomial degree $k \geq 0$, $\mathbb{P}_k(K)$ being the space of polynomials of global degree at most k on the element K. To overcome the ill-conditioning of elemental mass matrices for higher-order polynomials on high aspect ratio and curved elements we have choose a hierarchical and orthogonal set of shape functions defined in physical space. This set is obtained by using a modified Gram-Schmidt procedure, considering as a starting point a set of monomial functions of the same degree k.

Following Brezzi et al. [8], it is also convenient to introduce the jump and average trace operators, which on a generic internal face $F \in \mathcal{F}_h^i$ are defined as:

$$[\![q]\!] \stackrel{\text{def}}{=} q^+\mathbf{n}^+ + q^-\mathbf{n}^-, \qquad \{\cdot\} \stackrel{\text{def}}{=} \frac{(\cdot)^+ + (\cdot)^-}{2}, \tag{8}$$

where q denotes a generic scalar quantity and the average operator applies to scalars and vector quantities. By definition, $[\![q]\!]$ is a vector quantity. These definitions can be suitably extended to faces intersecting $\partial\Omega$ accounting for the weak imposition of boundary conditions.

The discrete counterpart of equation (5) for a generic element $K \in \mathcal{T}_h$ then reads:

$$\int_K \phi_h \frac{\partial \mathbf{u}_h}{\partial t} \, \mathrm{d}\mathbf{x} - \int_K \boldsymbol{\nabla}_h \phi_h \cdot \mathbf{F}(\mathbf{u}_h, \boldsymbol{\nabla}_h \mathbf{u}_h) \, \mathrm{d}\mathbf{x} + \int_{\partial K} \phi_h \mathbf{F}(\mathbf{u}_h|_K, \boldsymbol{\nabla}_h \mathbf{u}_h|_K) \cdot \mathbf{n} \, \mathrm{d}\sigma = \mathbf{0}. \quad (9)$$

Due to the discontinuous functional approximation, the flux function in the third term of equation (9) is not uniquely defined. Moreover, a consistent and stable discretization of viscous fluxes must account for the effect of interface discontinuities on the gradient $\boldsymbol{\nabla}_h \mathbf{u}_h$. Uniqueness of interface fluxes is achieved by introducing a suitably defined numerical flux which couples the solution in adjacent elements and ensures local conservation.

The inviscid $\widehat{\mathbf{F}}_c$ and viscous $\widehat{\mathbf{F}}_v$ parts of the numerical flux are treated independently. For the former, exact Riemann flux, approximate Roe flux–difference splitting or the approximate Van Leer flux-difference splitting as modified by Hänel [9] were used, while for the viscous flux discretization we adopted the BR2 scheme presented in[10, 11] and theoretically analyzed in [8, 12] (where it is referred to as BRMPS). The numerical viscous flux is given by:

$$\widehat{\mathbf{F}}_v \left(\mathbf{u}_h^{\pm}, (\boldsymbol{\nabla}_h \mathbf{u}_h + \eta_F \mathbf{r}_F(\llbracket \mathbf{u}_h \rrbracket))^{\pm} \right) \overset{\text{def}}{=} \{ \mathbf{F}_v \left(\mathbf{u}_h, \boldsymbol{\nabla}_h \mathbf{u}_h + \eta_F \mathbf{r}_F(\llbracket \mathbf{u}_h \rrbracket) \right) \}, \quad (10)$$

where, according to [8, 12], the penalty factor η_F must be greater than the number of faces of the elements. A very interesting feature of the outlined viscous flux discretization scheme is that it couples only the unknowns already coupled by the inviscid flux discretization scheme, irrespective of the degree of polynomial approximation of the solution.

Summing equation (9) over the elements and accounting for the above considerations, we obtain the DG formulation of problem (5), which then requires to find $u_{h_1}, \dots, u_{h_M} \in \Phi_h$ such that:

$$\begin{aligned} \int_{\Omega_h} \phi_h \frac{\partial \mathbf{u}_h}{\partial t} \, \mathrm{d}\mathbf{x} - \int_{\Omega_h} \boldsymbol{\nabla}_h \phi_h \cdot \left(\mathbf{F}_c \left(\mathbf{u}_h \right) + \mathbf{F}_v \left(\mathbf{u}_h, \boldsymbol{\nabla}_h \mathbf{u}_h + \mathbf{r} \left(\llbracket \mathbf{u}_h \rrbracket \right) \right) \right) \, \mathrm{d}\mathbf{x} \\ + \int_{\Gamma_h} \llbracket \phi_h \rrbracket \cdot \left(\widehat{\mathbf{F}}_c \left(\mathbf{u}_h^{\pm} \right) + \widehat{\mathbf{F}}_v \left(\mathbf{u}_h^{\pm}, (\boldsymbol{\nabla}_h \mathbf{u}_h + \eta_F \mathbf{r}_F(\llbracket \mathbf{u}_h \rrbracket))^{\pm} \right) \right) \, \mathrm{d}\sigma = \mathbf{0}, \end{aligned} \quad (11)$$

for all $\phi_h \in \Phi_h$. The lifting operator $\mathbf{r}_F$, which is assumed to act on the jumps of $\mathbf{u}_h$ componentwise, is defined as the solution of the following problem:

$$\int_{\Omega_h} \boldsymbol{\phi}_h \cdot \mathbf{r}_F \left(\mathbf{v} \right) \, \mathrm{d}\mathbf{x} = - \int_F \{ \boldsymbol{\phi}_h \} \cdot \mathbf{v} \, \mathrm{d}\sigma, \forall \boldsymbol{\phi}_h \in \left[\Phi_h \right]^N, \; \mathbf{v} \in \left[L^1 \left(e \right) \right]^N, \quad (12)$$

and the function $\mathbf{r}$ is related to $\mathbf{r}_F$ by the equation:

$$\mathbf{r}(\mathbf{v}) \overset{\text{def}}{=} \sum_{F \in \mathcal{F}_h} \mathbf{r}_F(\mathbf{v}). \quad (13)$$

At the boundary of the domain, the numerical flux function appearing in equation (11) must be consistent with the boundary conditions of the problem. In practice, this is accomplished by properly defining a boundary state which accounts for the boundary data and, together with the internal state, allows to compute the numerical fluxes and the lifting operator on the portion Γ_h^b of the boundary Γ_h, see [10, 13].

3 Time integration

Numerical integration of Eq. (11) by means of suitable Gauss quadrature rules leads to a system of nonlinear ODEs that can be written as

$$\mathbf{M}\frac{\mathrm{d}\mathbf{U}}{\mathrm{d}t} + \mathbf{R}\left(\mathbf{U}\right) = \mathbf{0}, \tag{14}$$

where $\mathbf{U}$ is the vector of the degrees of freedom, $\mathbf{R}\left(\mathbf{U}\right)$ is the residuals vector, and $\mathbf{M}$ is the block diagonal mass matrix, which is, in our implementation, equal to the identity matrix due to the choice of orthonormalized shape functions. It is worth noting that the degrees of freedoms (DOFs) are related to the solution via the relation

$$u_{h,i} = U_{i,j}\phi_j$$

for all the elements $K \in \mathcal{T}_h$, where $i = 1, ..., m$, being m the number of conservative variables, and $j = 1, ..., N^K_{dof}$, being N^K_{dof} the number of degrees of freedom on the element K. In this paper we integrate in time the system (14) by using the explicit three stage, order three, strong stability–preserving Runge–Kutta scheme of Gottlieb et al. [14]. The time integration of Eq. (14) by means of this low-storage scheme can be written as

$$\begin{aligned}
\mathbf{U}^{(0)} &= \mathbf{U}^n, \\
\mathbf{U}^{(i)} &= \sum_{k=0}^{i-1}\left(\alpha_{i,k}\mathbf{U}^{(k)} + \Delta t\beta_{i,k}\mathbf{R}(\mathbf{U}^{(k)})\right), \qquad i = 1, ..., s \\
\mathbf{U}^{n+1} &= \mathbf{U}^{(s)}.
\end{aligned} \tag{15}$$

where s is the number of stages, $\alpha_{i,k}, \beta_{i,k}$ are the coefficients of the scheme [14]. For the sake of convenience, the dependence of the residuals vector by the independent variables $\mathbf{U}$ will be omitted in the rest of the paper, moreover $\mathbf{R}$ will be splitted into the volumes, $\mathbf{R}_K$ with $K \in \mathcal{T}_h$, and the surfaces, $\mathbf{R}_F$ with $F \in \mathcal{F}_h$, contributions. The entries of $\mathbf{R}_K$ and $\mathbf{R}_F$ can be found by comparison with Eq. (11).

4 Parallel OpenMP implementation

In order to implement the hybrid parallelization strategy, based upon the MPI and OpenMP paradigms, it is first convenient to study an efficient and practical use of the OpenMP within our existing algorithm. This can be done by considering the main characteristics of a modal DG framework. In fact, it is typically convenient to arrange by elements the main arrays $\mathbf{U}$ and $\mathbf{R}$ and to perform the evaluation of the latter vector according to two main loops. In the first one, performed over the ne elements, all the volume contributions, $\mathbf{R}_K$, are evaluated. In the second, performed over the faces $F \in \mathcal{F}_h$, the numerical fluxes contributions $\mathbf{R}_F$ are handled. Again, the latter operation can be splitted in two parts: the first covers all the internal faces, $\mathbf{R}_{F_i}$ with $F \in \mathcal{F}^i_h$, the second takes in account all the boundaries faces, $\mathbf{R}_{F_b}$ with $F \in \mathcal{F}^b_h$.

In the contest of OpenMP, the evaluation of the $\mathbf{R}_K$ contribution to the residual vector can be safely parallelized, for example using the Fortran `!$OMP do` work sharing directive, without data race condition problems. The use of this straightforward strategy is not possible for the loop over the faces, since for each face the residuals of both the K^- and K^+ elements are simultaneously updated. Performing this loop in parallel require special attention, since two threads may overwrite, at once, the same memory location of the $\mathbf{R}$ vector.

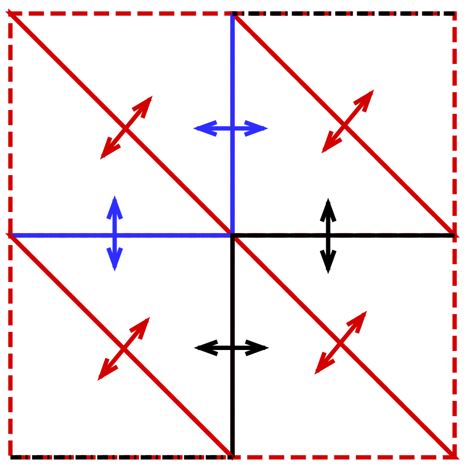


Figure 1: Representation of the colouring algorithm

To solve this issue we adopted a colouring strategy. The colouring procedure is based on a naive and simple to code algorithm similar to that used in [15, 16]. Here the main difference is that the colouring procedure is applied to the faces and not to the elements. The key idea is that one element cannot have more than one face of the same colour. The algorithm is described as follows: when the face F has to be coloured, one must check the already assigned colours of the neighbour faces. The scope is to assign a different colour to F. If there is not a available colour in the current palette the number of colours is increased by one and this new colour is used for F. Note that the neighbour faces correspond to all the faces owned by the two elements connected by the face F. Once that all the faces are coloured the contribution of numerical fluxes to the residual vector can be performed in parallel for all the faces of the same colour, ensuring that the data race condition is avoided. In Figure 1 the $\mathbf{R}_{F_i}$ components due to the four red internal faces can be summed concurrently (in the figure this operation is represented by an arrow) to the $\mathbf{R}$ global vector, the same consideration holds true for the two black and for the two blue faces. It is clear that this algorithm is not optimized to obtain a group of colours characterized by an equal number of faces. However, it is fast and it performs well as will be shown in the remaining of the paper. Although it is evident that the colours distribution change with the initial ordering of the faces, as well as with grid reordering, we verified that the overall parallel performance is not significantly affected. Moreover, as pointed out in [16], it is possible to add a a posteriori phase to the algorithm to balance the number of faces for each colour.

Regarding the boundary conditions, the same algorithm is used. Indeed, in this circumstance, almost all the boundary faces have the same colour and only if one element owns more than one boundary face a further colour is needed. Even in this case all the boundary of the same colour can be safely evaluated in parallel, see Figure 1 in which the boundaries are depicted with dashed lines. Finally, it is worth mentioning that the selected algorithm is well suited to deal with accelerators, such as GPUs [16] or the Xeon Phi coprocessors in its offload mode, and that it can be easily implemented in the context of an MPI solver resulting in a hybrid strategy.

Hybrid MPI/OpenMP implementation can also take advantage of possibly non–blocking communications which are overlapped with computations. More in detail, the $\mathbf{R}_{F_i}$ evaluation and the exchange of messages required for updating the ghost elements DOFs are simultaneously performed. Note that for the diffusive volume terms our DG implementation uses the

BR2 scheme [1], which relies on the lifting operator $\mathbf{r}\left(\llbracket \mathbf{u}_h \rrbracket\right)$, that depends on the jumps of the variables at the faces. It is more computationally convenient to evaluate these jumps in the loop over the faces and to compute only later the $\mathbf{R}_K$ residual contributions. Due to this implementation it is clearly not possible to overlap the last task with the MPI communications in the viscous case.

5 The Taylor Green Vortex problem

The Taylor Green Vortex problem is a canonical test case used for studying the mechanism upon which large scales vortices interact with each other producing turbulence decay and thus allowing the study of the energy dissipation processes. Since the geometry, consisting on a periodic cube with sides of length $2\pi L$, is considerably simple, it has been used extensively for testing the effects of numerical methods, grid resolution and turbulence modelling strategies like DNS and LES on the simulation of turbulent flows.

The problem consists of a cubic volume of fluid that contains a smooth initial distribution of vorticity, given by the following initial field

$$u = V_0 \sin\left(\frac{x}{L}\right) \cos\left(\frac{y}{L}\right) \cos\left(\frac{z}{L}\right), \tag{16}$$

$$v = -V_0 \cos\left(\frac{x}{L}\right) \sin\left(\frac{y}{L}\right) \cos\left(\frac{z}{L}\right), \tag{17}$$

$$w = 0, \tag{18}$$

$$\rho = \rho_0, \tag{19}$$

$$p = p_0 + \frac{1}{16}\left(\cos\left(\frac{x}{L}\right) + \cos\left(\frac{y}{L}\right)\right)\left(\cos\left(\frac{z}{L}\right) + 2\right). \tag{20}$$

The initial vortices start interacting together, resulting in a transition to turbulence and in the following turbulence decay for sufficiently high Reynolds numbers. As there is no forcing term on the equations, the flow field will slowly dissipate all the initial kinetic energy and it will come to rest.

The flow conditions are those used in [17]. The Mach number M is equal to 0.1, resulting in a nearly–incompressible simulation, and the Prandtl number is 0.72. Two different Reynolds numbers have been employed, $Re = \rho_0 V_0 L/\mu = 800$ and 1600. Both the Reynolds numbers are high enough to make the solution unstable and to produce a nearly isotropic turbulent decay. In order to monitor the development of the turbulent flow over time, two quantities have been employed. The first one is the time derivative of the kinetic energy defined as

$$k \overset{\text{def}}{=} \frac{1}{16\pi^3 \rho_0} \int_\Omega \rho u_k u_k \,\mathrm{d}\mathbf{x}. \tag{21}$$

The second one is the theoretical dissipation rate, evaluated as

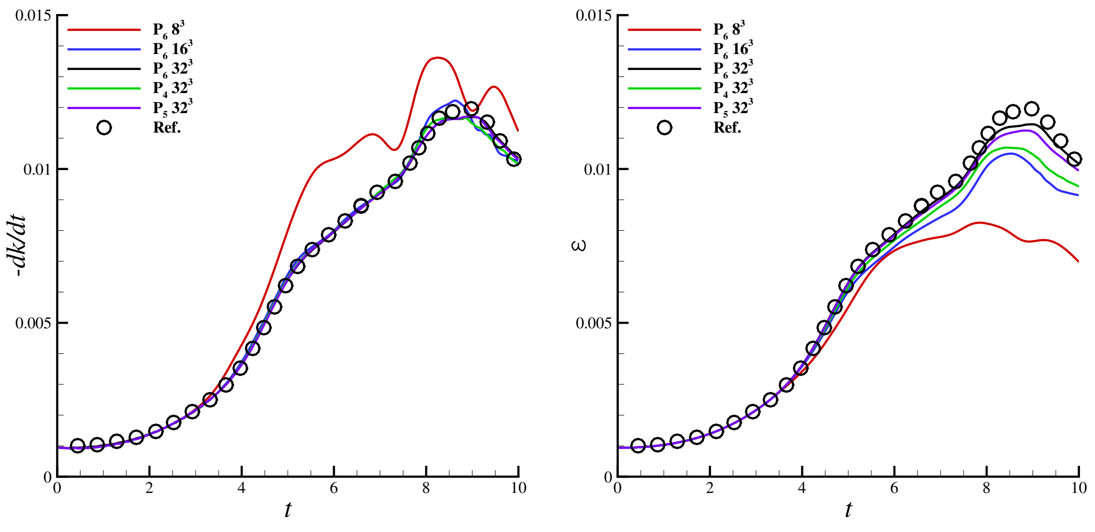
$$\varepsilon \overset{\text{def}}{=} \frac{2\mu}{8\pi^3 \rho_0} \int_\Omega \frac{\partial u_i}{\partial x_j}\frac{\partial u_i}{\partial x_j} \,\mathrm{d}\mathbf{x}. \tag{22}$$

Theoretically, in the incompressible limit, the following relation holds

$$-\frac{\mathrm{d}k}{\mathrm{d}t} = \varepsilon.$$

The numerical difference between these two quantities is thus somehow connected to the quality of the simulation and its numerical dissipation.

The solutions were advanced with a time step size corresponding to a maximum CFL number of 0.1, for both the $\mathbb{P}_4$ and the $\mathbb{P}_5$ approximations, or to 0.08, for $\mathbb{P}_6$. These values results in negligible time discretization errors. Figure 2 shows the kinetic energy time derivative and the theoretical dissipation rate for the case at $Re = 800$ for different grid sizes as well as for different order of polynomial approximation. The figure highlights the increment in resolution obtained increasing both the order of accuracy of the solution and the grid dimension. It is clear from Figure 2(a) that the evolution in time t, non–dimensionalized by V_0 and L, of the kinetic energy time derivative is not captured by the coarsest grid (8^3 elements). However, the solutions are in good agreement with the reference DNS data (Brachet et al. [18], digitalized from Ref. [17]) starting from the 16^3 $\mathbb{P}_6$ case to the more resolved cases. Only a small difference can be noted between the $\mathbb{P}_5$ and the $\mathbb{P}_6$ solutions obtained on the finest grid. The largest value of this difference is close to $t = 8.5$, when the dissipation reaches its maximum value. Figure 2(b) depicts the theoretical dissipation rate evaluated via Eq.(22). In this case, slightly larger differences between ε and reference data can be observed. This result suggests that for the most of



(a) Kinetic energy k time derivative (b) Theoretical dissipation rate, ε

Figure 2: Taylor Green vortex at $Re = 800$. Results for different numerical resolutions.

the cases our computations under–resolve the turbulent scales. It is worth noting that the 16^3 $\mathbb{P}_6$ and 32^3 $\mathbb{P}_4$ solutions produce nearly the same numerical dissipation over time. However, they are obtained with a quite large difference of DOFs, which are approximately equal to 70^3 for the former and to 105^3 for the latter case. This behaviour shows the benefits of using an increased order of polynomial approximation instead of refining the mesh. Figure 3 shows the $\lambda_2 = -1.5$ iso–surface plot as reported in Ref. [17]. The resulting flow field resembles the reference one, and as the calculation is refined the turbulent structures become more clean and defined, especially for the 32^3 $\mathbb{P}_6$ case.

The effects of using different convective fluxes can be observed in Figure 4. The Roe approximate solver produces a negligible difference with respect to the exact Riemann solver, that exploits the numerical iterative procedure of Gottlieb et al. [19]. The Hänel flux splitting is revealed to be less accurate, as the theoretical dissipation rate, see Figure 4(b), is lower if compared to the reference data and to the results obtained with the other fluxes.

Figure 5 shows the results for $Re = 1600$, with the same grids and polynomial orders of

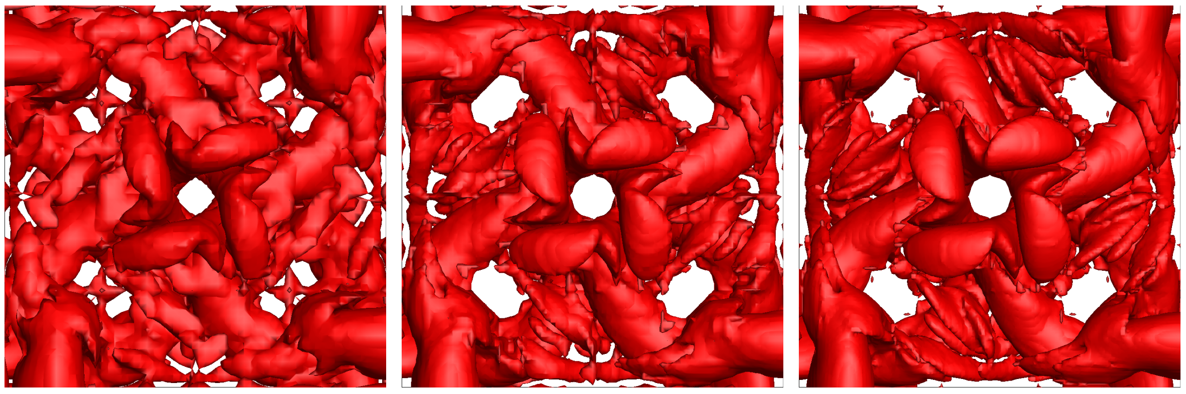
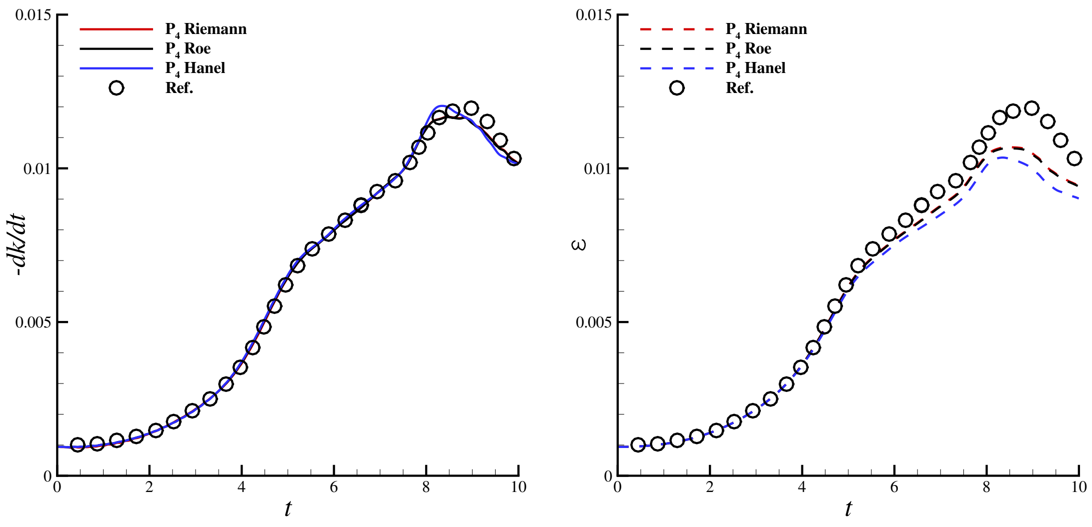

(a) 16^3 $\mathbb{P}_6$ (b) 32^3 $\mathbb{P}_4$ (c) 32^3 $\mathbb{P}_6$

Figure 3: Taylor–Green vortex at $Re = 800$. $\lambda_2 = -1.5$ iso–surface at $t = 8.5$.



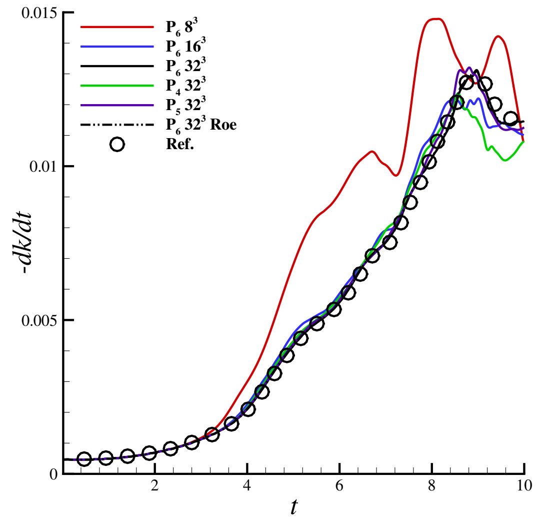
(a) Kinetic energy k time derivative (b) Theoretical dissipation rate, ε

Figure 4: Taylor Green vortex at $Re = 800$. Results for different numerical fluxes at 32^3 $\mathbb{P}_4$.

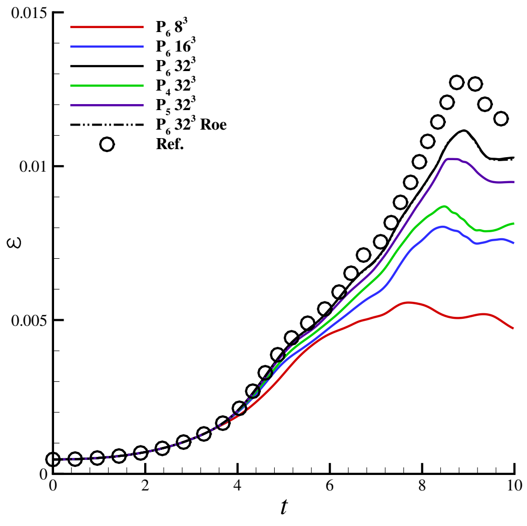
the previous test case. Since the smallest scales of turbulent decreases in size as the Reynolds number increases, the turbulence is clearly under–resolved. Figure 5(a) shows that the kinetic energy dissipation rate, for the test case 32^3 $\mathbb{P}_6$, is in good agreement with the reference DNS 512^3 pseudo–spectral solution [20]. Concerning the theoretical dissipation rate, see Figure 5(b), the difference from the reference data grows, showing that the numerical setup for this Reynolds number filters part of the smallest turbulent scales. Moreover, the use of the Roe numerical flux, employed for the finest test case, produces negligible difference in dissipation from the exact Riemann solver, but reduces of about 20% the computational time, see Table 1.

Figure 6 shows the $\lambda_2 = -1.5$ iso–surface coloured by the vorticity magnitude. It appears clear that as the resolution is increased the turbulent structures become smaller and more defined.

Figure 7 reports the sum $-\,\mathrm{d}k/\,\mathrm{d}t - \varepsilon$, which ideally should be zero. The differences from zero are related to the lack of numerical resolution. For both the solutions presented, as the polynomial order as well as the grid resolution increases, the imbalance tends to diminish. A

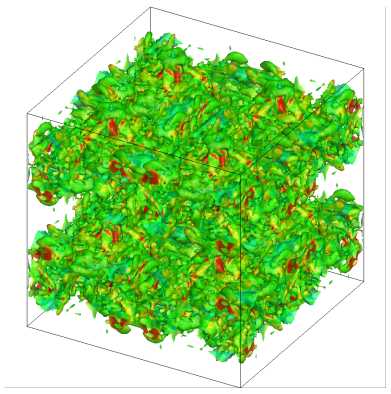


(a) Kinetic energy k time derivative

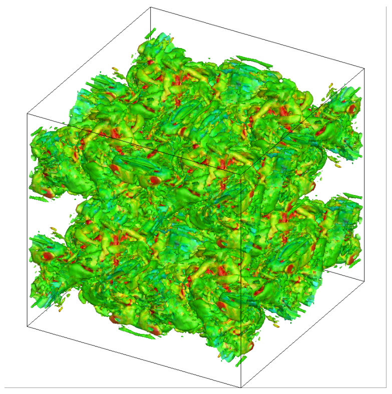


(b) Theoretical dissipation rate, ε

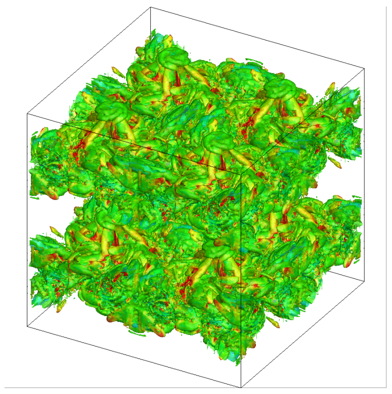
Figure 5: Taylor Green vortex at $Re = 1600$. Results for different numerical resolutions. Exact Riemann solver employed unless otherwise stated.



(a) 16^3 $\mathbb{P}_6$

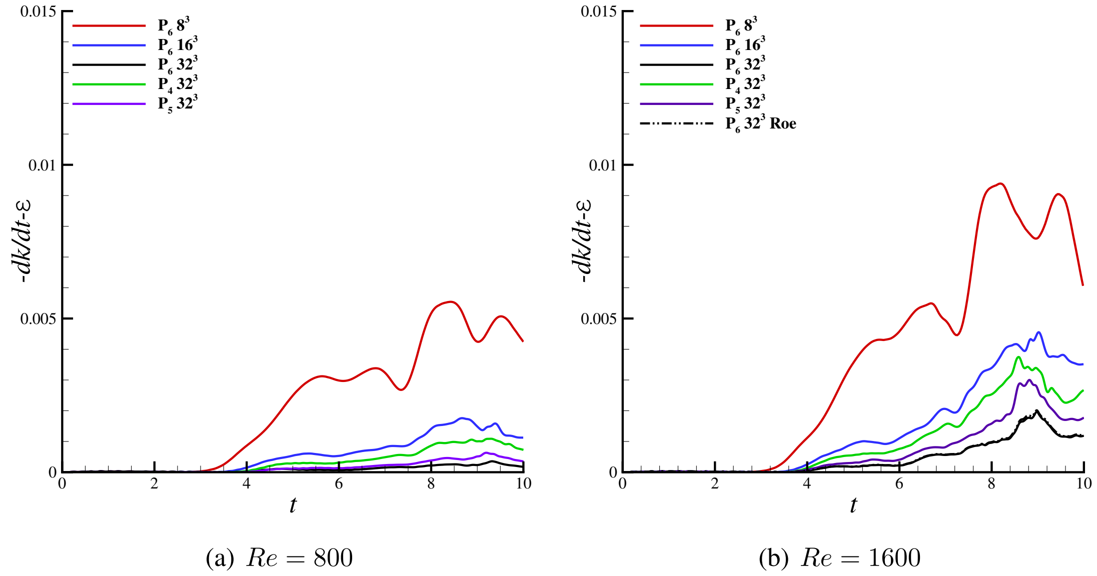


(b) 32^3 $\mathbb{P}_4$



(c) 32^3 $\mathbb{P}_6$

Figure 6: Taylor–Green vortex at $Re = 1600$. $\lambda_2 = -1.5$ iso–surface at $t = 10$. Iso–surface coloured by non–dimensional vorticity magnitude.



(a) $Re = 800$ (b) $Re = 1600$

Figure 7: Taylor Green vortex. $-\varepsilon - \ \mathrm{d}k/\ \mathrm{d}t$ imbalance as function of non–dimensional time.

peak value around $t = 8.5$ is produced, due to the fact that the dissipation reaches a maximum and thus the turbulent scales produced in the flow field are the smallest. This clearly stresses the resolution properties of the schemes. As shown in Figure 7(a), the 32^3 $\mathbb{P}_6$ computation is able to solve almost completely the turbulent structures at $Re = 800$. However, the same numerical discretization filters the smallest scales as the Reynolds is increased, see Figure 7(b). Additionally, the Roe flux–difference splitting confirms its benefits over the exact Riemann solver, not affecting the numerical resolution but increasing the efficiency of the computation.

Case	DOFs	$\Delta\varepsilon_1$	$\Delta\varepsilon_2$	Work Units
8^3 $\mathbb{P}_6$	35^3	$3.2 \cdot 10^{-1}$	$7.3 \cdot 10^{-1}$	$3.0 \cdot 10^3$
16^3 $\mathbb{P}_6$	70^3	$8.8 \cdot 10^{-2}$	$3.5 \cdot 10^{-1}$	$5.3 \cdot 10^4$
32^3 $\mathbb{P}_4$	105^3	$2.1 \cdot 10^{-1}$	$2.9 \cdot 10^{-1}$	$2.1 \cdot 10^5$
32^3 $\mathbb{P}_5$	122^3	$6.1 \cdot 10^{-2}$	$2.3 \cdot 10^-1$	$5.6 \cdot 10^5$
32^3 $\mathbb{P}_6$	140^3	$2.5 \cdot 10^{-2}$	$1.5 \cdot 10^{-1}$	$9.3 \cdot 10^5$
32^3 $\mathbb{P}_6$, Roe	140^3	$2.5 \cdot 10^{-2}$	$1.5 \cdot 10^{-1}$	$7.7 \cdot 10^5$

Table 1: Taylor Green vortex at $Re = 1600$. Error norms and work units for the hybrid MPI/OpenMP solver

The computational efforts, expressed in work units as defined in the 1st International Workshop on High-Order CFD Methods [21], for the $Re = 1600$ case are reported in Table 1 together with the following error norms

$$\Delta\varepsilon_1 = \frac{||dk/dt - (dk/dt)_{ref}||_{L\infty}}{\max\left(-(dk/dt)_{ref}\right)},$$

$$\Delta\varepsilon_2 = \frac{||-\epsilon - dk/dt||_{L\infty}}{\max(-(dk/dt)_{ref})},$$

where the subscript ref indicates the Van Ress et al. [20] solution taken as reference in that Workshop also. It is worth noting that, since we use a polynomial approximation of global degree at most of k for a $\mathbb{P}_k$ solution, our finest simulation contains only about 140^3 DOFs for each of the physical variable. Clearly the theoretical order of convergence is not reached, but it should be considered that the reported results are somehow under–resolved. Anyway, if compared with the results reported in [21], our data appear satisfactory both in terms of solution quality and computational efficiency.

6 Parallel performance of the Hybrid OpenMP/MPI

To obtain reliable indications about the parallel behaviour of the proposed hybrid implementation we report benchmarks results obtained on different multi–core machines, including clusters. The characteristics of each of these computational platforms (hardware and software) have a relevant influence on the parallel performance, thus a brief description of these architecture will precede each numerical experiment. If not otherwise stated, the TGV problem has been considered here to asses the performances of the the different parallel programming paradigms. The timings were measured advancing the solution for 10 Runge–Kutta steps, which ensures the independence of the results from eventual delay due to possible background processes of the operative system. We chose three different grid sizes, 8^3, 16^3 and 32^3 as well as three different polynomial orders, $\mathbb{P}_1$, $\mathbb{P}_3$ and $\mathbb{P}_5$.

To get indications reflecting the real usage of the solver, they were used in a completely standard configuration, for instance even the $||\mathbf{R}||_{L_2}$ and $||\mathbf{R}||_{L_\infty}$ values were written to a file at the beginning of each time step. This task is performed exclusively to allow the user control of the computation status and it is not strictly required by the DG method. To maximize the performance we used in all the cases the OpenMP capability to bind threads to cores, while, as regards the threads affinity, generally we observed that the way in which the threads are assigned to the cores is not particularly relevant. The only exception is when a core pair of the AMD 6276 Opteron CPU is employed, as it will be explained in the next sections.

For the sake of compactness in this paper we will use the following notation to describe a parallel setting of a particular simulation: an hybrid MPI/OpenMP computation will be denoted by a triple of positive integers (n, m, t), where n is the number of the employed nodes, m represents the MPI processes running in each of this node and t reflects the OpenMP threads activated for each of the MPI process. Clearly the product of these three numbers indicates the total number of threads used, and usually this number is at most equal to the number of the available cores. For a pure MPI run we will use only the couple of numbers (n, m), since obviously t is always equal to 1. Finally, note that in this work all the computational grids were partitioned by means of the Metis [22] library.

6.1 Results on the 1st multi–core platform

The first system consists of two nodes interconnected by a point to point Infiniband network. Each node is equipped with two AMD Opteron 6276 processors, each with sixteen cores working at a fixed 2.3 GHz frequency, since the application power management (APM) of the operating system, Linux Debian 3.2.32-1, was disabled. Moreover even the AMD Turbo Core technology was also turned off since it can dynamically increase up to 3.2 GHz the operating frequency of a core. It is obvious that using these technologies it is not possible to produce reliable and repeatable indications about the OpenMP scalability.

The sixteen cores AMD 6276 CPU is a multi–chip module built by two eight–cores CPU die placed on a single die package (interconnected by a HyperTransport link). Each CPU in the package has its own memory controller, which is shared by its 8 cores. This configuration leads to 4 NUMA regions, see [23]. Another relevant aspect of this hardware platform is that the cores are organized in core pair, sharing some components such as the floating point units and the L2 cache. Due to this hardware configuration the maximum overall performance of one socket should be not equal to sixteen times the single core performance. Indeed, we have verified that the usage of both the cores of the pair, running two identical instances of the serial DG solver, a significant performance reduction can be observed. In fact, the execution time becomes larger, approximately equal to 1.25 times the standard wall clock time. Moreover, this performance degradation is not visible if cores of different pairs are employed in the computation. As a direct consequence, the theoretical maximum speed–up achievable using all the 32 cores is limited to about 25.6, corresponding to an apparent parallel efficiency of 80%.

The code is compiled, and optimized following the AMD indications found in [24], with the Open64 compiler suite. We verified that this choice, on our specific hardware, increases extensively the performances of our solver. The MVAPICH-2.2.1 library, which is an open source library suited for Infiniband networks and derived from the MPICH-3 library, was used for MPI communications. This library uses shared memory channels for the inter–node communications, thus the message passing between cores sharing the memory is faster then message passing between nodes.

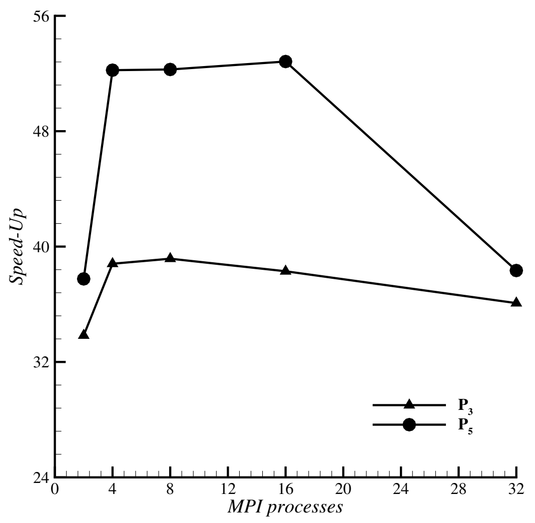


Figure 8: Hybrid MPI/OpenMP solver, Speed–up using 32 AMD Opteron 6276 cores on the 32^3 grid.

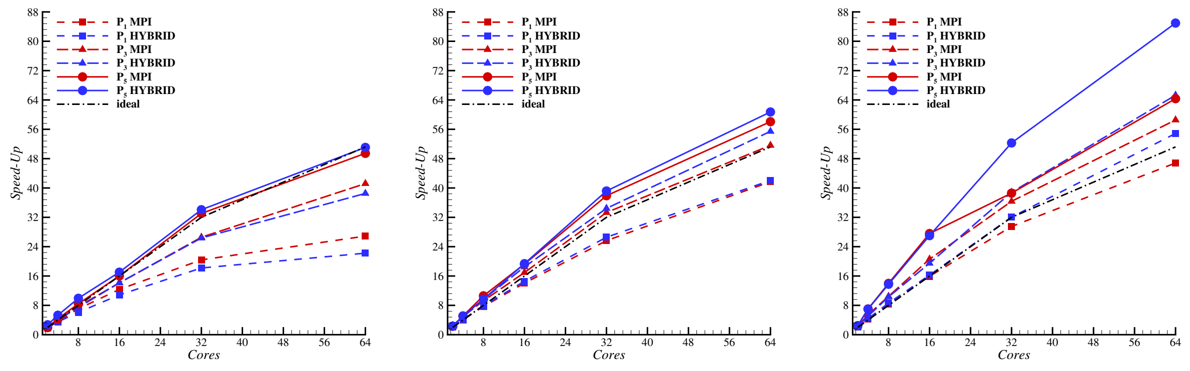
According to Guo et al. [25] we firstly investigated how the performance is affected by the subdivision of MPI partitions and OpenMP threads. In fact, those authors observed a significant performance reduction using 16 or 32 threads on a node of the HECToR Cray XE6 system, which consists of exactly two AMD Opteron 6276 CPUs. In their work this behaviour was ascribed to the latencies and the bandwidth reduction related to the presence of the 4 NUMA regions. To obtain a better parallel efficiency the authors used 4 MPI process per node and 8 OpenMP threads per each MPI process. The same idea has been applied to determine which configuration is able to exploit better performance of the hybrid code. For the 32^3 grid a comparison between different strategies have been performed and reported in Figure 8. It should be noted that the tests have been done using only the half of the cores available on the two nodes to avoid the use of the core pairs and the intrinsic computational penalty. It can be seen clearly that the Speed–Up grows when the total MPI partitions are between 4 and 16 reaching a plateau.

This observation is confirmed both for the $\mathbb{P}_3$ case and for the $\mathbb{P}_5$ as well. From those evidences it can be deduced that the optimal use of the hybrid algorithm coincides with having a number of MPI partitions comparable to the number of NUMA regions. The strong scalability curves of Figure 9 has been obtained following this approach. Table 2 reports the resulting optimal subdivision between the OpenMP threads and MPI processes, using the compact notation introduced above.

Cores	MPI	Hybrid MPI/OpenMP
1	$(1,1)$	$(1,1,1)$
2	$(1,2)$	$(1,2,1)$
4	$(1,4)$	$(1,4,1)$
8	$(1,8)$	$(1,4,2)$
16	$(1,16)$	$(1,4,4)$
32	$(2,16)$	$(2,4,4)$
64	$(2,32)$	$(2,4,8)$

Table 2: Subdivision between MPI processes and OpenMP threads on the AMD Opteron 6276 system

In Figure 9 the Speed–Up trends for the three different grids, increasing the polynomial order from $\mathbb{P}_1$ to $\mathbb{P}_5$, are reported. Figure 9(a), which is related to the coarsest (8^3) grid, clearly highlights that the parallel efficiency raises with the polynomial order and that it reaches the ideal scaling at $\mathbb{P}_5$. Note that the pure MPI implementation is more efficient at the lower order approximation, but at $\mathbb{P}_5$ the hybrid version is slightly more convenient.



(a) 8^3 grid (b) 16^3 grid (c) 32^3 grid

Figure 9: Performance of the hybrid MPI/OpenMP and the pure MPI solvers on four AMD Opteron 6276 CPUs for different grid sizes and $\mathbb{P}_1$, $\mathbb{P}_3$ and $\mathbb{P}_5$.

For the 16^3 grid, see 9(b), the pure MPI code version performs considerably well, especially at $\mathbb{P}_5$ when the Speed–Up curve is considerably above the ideal scaling law. This behaviour is partially due to the poor performance of the serial version of the code in dealing with a large memory allocation which is typical of the high–order approximation. The hybrid curve is in this circumstance always above the one obtained using only the MPI parallelization strategy. Those observations and trends are confirmed increasing the grid size, see Figure 9(c). The results show that at highest computational loads the Speed–Up can be highly super–linear. It is worth noting that with the 32^3 grid, and 5th degree of polynomial approximation, the amount of memory required exceeds 29Gb, and probably the TLB misses (Translation Lookaside Buffer) reduces the application performance. In this case the *HugePage* support in the Linux kernel should be enabled to limit this overhead. Moreover, the memory requirement exceeds the available RAM of each NUMA region (the total amount of RAM for each node is 64Gb) and the memory bandwidth and latency could be the bottlenecks. Just by partitioning the execution in two MPI processes, since every process deals with the half of the total memory, the parallel efficiency is highly super–linear, producing an efficiency of 120%. With 4 MPI processes the efficiency still raises up to 170%, and this gain is affecting all the other numerical tests. However only the hybrid implementation is able to retain this performance advantage (over the serial application) when 32 and 64 cores are used. Using all the cores this behaviour results in a performance improvement of about the 38%.

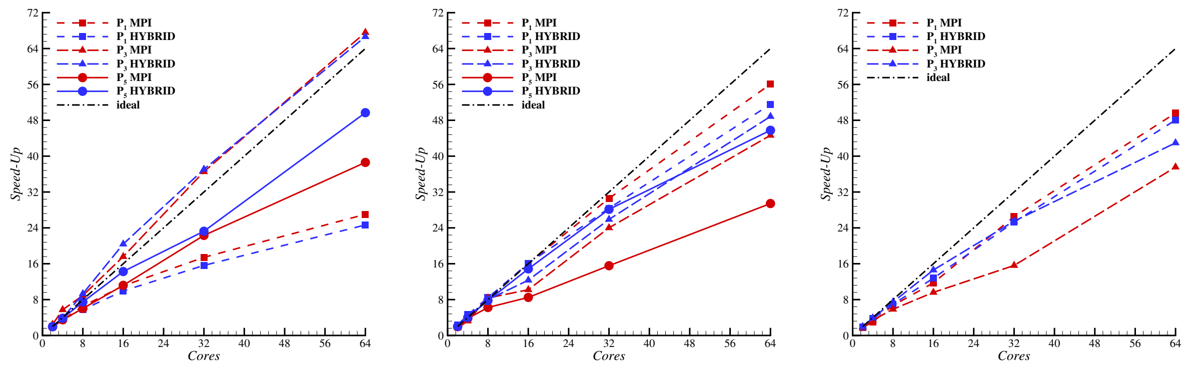
6.2 Results on the 2nd multi–core platform

The second cluster is built by 8 nodes interconnected by a gibabit ethernet that takes advantage of a single switch. This network serves also the Network File System (NFS). Due to these hardware characteristics the system is expected to scale poorly, especially when all the available 64 cores are involved in a parallel job. Each node consists of two NUMA regions equipped by a quad–cores AMD Opteron 2378 CPUs, running at 2.4 GHz. The operating system is the same employed in the first machine, while in this circumstance we used the GNU compilers 4.7.2 and the MPICH–3.1.4 library.

The strong scalability curves are reported in Figure 10. Three different grid sizes, 8^3, 16^3 and 32^3 have been employed for the tests. For the first two grids, $\mathbb{P}_1$, $\mathbb{P}_3$ and $\mathbb{P}_5$ polynomial approximations have been used, while for the finest one, the scalability has been performed only using $\mathbb{P}_1$ and $\mathbb{P}_3$ polynomial approximation, due to the limitation in memory for running the scalar version of the $\mathbb{P}_5$ case. Despite this platform produced quite counterintuitive results (the Speed–Up does not always increase raising the polynomial order and/or the grid size) the hybrid OpenMP/MPI algorithm produces a remarkable improvement of the performance over the pure MPI approach, especially as the order of polynomial approximation grows. The advantage is above 15% as far as cases of considerable computational load are considered. Also with this machine super–linear scalability effects, due to cache efficiency, can be observed in some cases, mainly at 8^3 $\mathbb{P}_3$. The experimented subdivision between MPI partitions and OpenMP threads is reported in Table 3.

Cores	MPI	Hybrid MPI/OpenMP
1	$(1,1)$	$(1,1,1)$
2	$(1,2)$	$(1,2,1)$
4	$(1,4)$	$(1,2,2)$
8	$(1,8)$	$(1,2,4)$
16	$(2,8)$	$(2,2,4)$
32	$(4,8)$	$(4,2,4)$
64	$(8,8)$	$(8,2,4)$

Table 3: Subdivision between MPI processes and OpenMP threads on the AMD Opteron 2378 system



(a) 8^3 grid (b) 16^3 grid (c) 32^3 grid

Figure 10: Performance of the hybrid MPI/OpenMP and the pure MPI solvers on sixteen AMD Opteron 2378 CPUs for different grid sizes and $\mathbb{P}_1$, $\mathbb{P}_3$ and $\mathbb{P}_5$.

6.3 Results on the BlueGene/Q platform

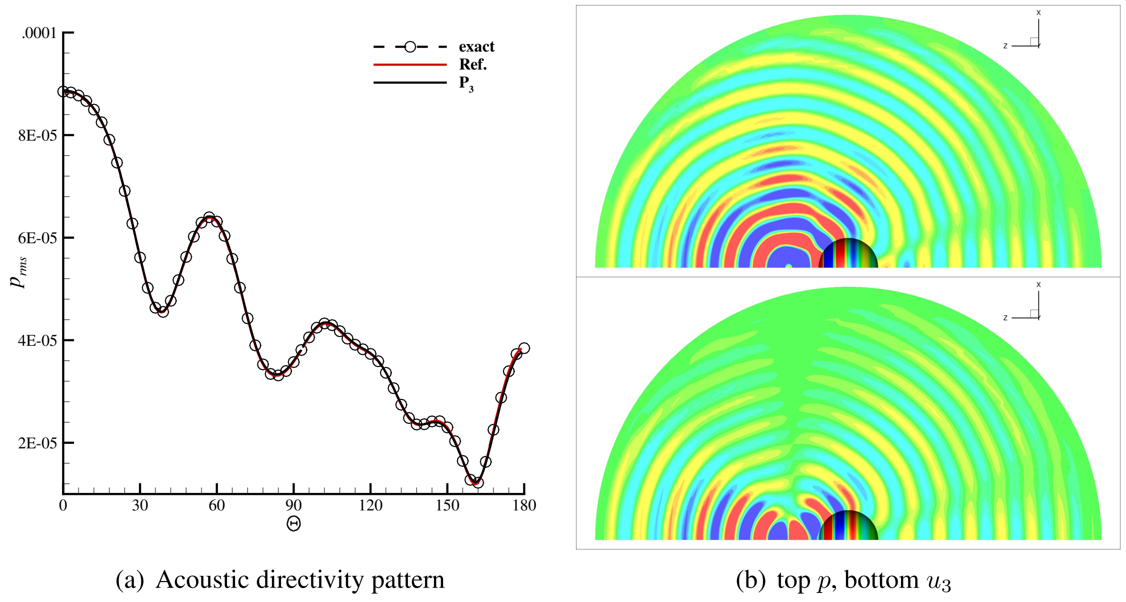
Although we still have to verify the performances of the hybrid Navier–Stokes DG solver on large HPC facilities, we already experimented the CAA (Computational Aeroacoustics) version of the current code on the IBM–BlueGene/Q platform. This machine, named FERMI, is available at CINECA, and it is characterized by a massively parallel architecture consisting of 10.240 nodes using sixteen cores of the IBM PowerPC A2 1.6 GHz chip. These cores are able to execute concurrently up to 4 hardware threads and better performance is expected exploiting more than one thread for each core. Each node is characterized by a symmetric access to all the memory, resulting in a Uniform Memory Access (UMA) machine.

The Problem 4 of Category 1 of Second Computational Aeroacoustics (CAA) Workshop on Benchmark Problems [26] was considered as test case. The problem of the acoustic scatter from a rigid sphere has been computed by solving the Linearized Euler Equations (LEE). The acoustic perturbations are generated by a spatially distributed acoustic source and the mean flow is at rest. The computational domain has been discretized using an unstructured grid consisting of 89628 hexahedral elements and the solution was computed using a $\mathbb{P}_3$ polynomial approximation.

Figure 11(a) shows that the computed acoustic directivity pattern compares very favourably with the analytical solution derived by Morris, [27], while Figure 11(b) presents a couple of pictures of the p and u_3 contours that help clarifying the solution structure.

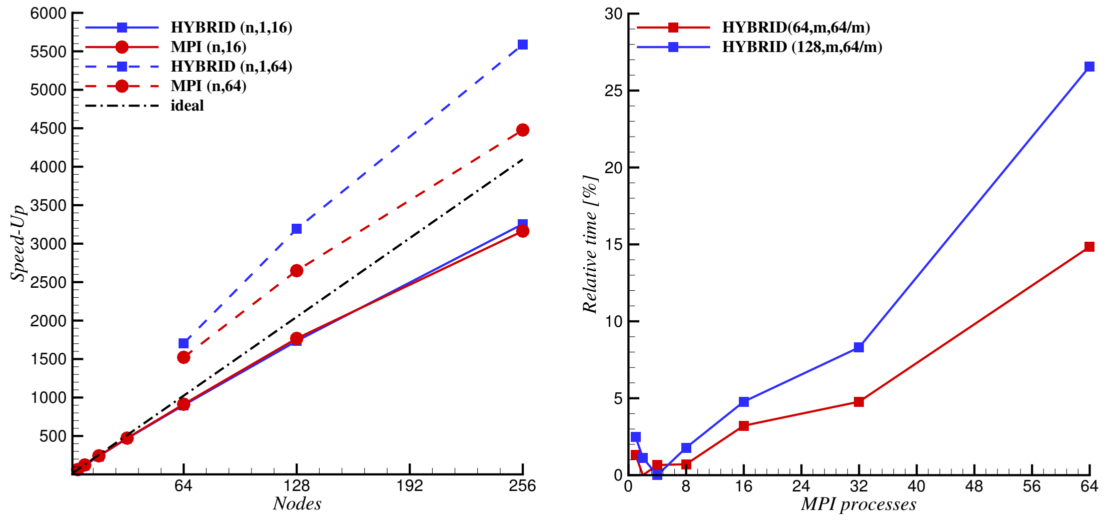
As regards the parallel performance, Figure 12(a) reports the measured Speed-Up. Using up to 16 threads, equal to the number of the cores, for each node we obtain good parallel performances adopting both the approaches. The hybrid MPI/OpenMP performs slightly better, for example its parallel efficiency at 4096 cores, corresponding to 22 elements per core, is 83.5% against the 77.2% pure MPI value. However, with 64 threads for node, 4 hardware threads for each core, the parallel performance of the hybrid code strongly outperforms that of the pure MPI one. This gain increases, at a fixed grid size, raising the number of the employed computational nodes. When using 16384 threads each process deals with an average value of 5.5 grid cells.

Figure 12(b) shows that the relative wall clock time, with respect to the best configuration, using all the 64 hardware threads and varying the combination between the MPI processes and the OpenMP threads. It comes clear that with this UMA platform the better performance is obtained by using a small number of MPI process for each node.



(a) Acoustic directivity pattern (b) top p, bottom u_3

Figure 11: Acoustic scattering from a sphere. $\mathbb{P}_3$ solution, Ref. solution is from [28], exact solution is due to Morris [27]



(a) Strong scalability (b) $(n, m, 64/m)$ relative time

Figure 12: Acoustic scattering from a sphere. Performance of the hybrid MPI/OpenMP and pure MPI implementations on the BlueGene/Q platform.

7 Conclusions

In this work we have considered two possibly OpenMP and hybrid OpenMP/MPI implementations of a high–order DG solver for CFD and CAA applications. OpenMP-based parallelism for shared memory architectures is usually expected to give large gains when compared to pure MPI implementations on massively parallel computers. Although the coding effort required by an OpenMP parallelization is quite small, obtaining an efficient solver and clear usage guidelines (optimal load repartition among OpenMP and MPI processes) is not trivial and many issues has to be considered. Here, a satisfactory parallel efficiency has been obtained using a simple yet effective colouring algorithm, tested on several problems.

Numerical experiments indicate that our OpenMP/MPI implementation improves the parallel efficiency of the solver on clusters made of largely multi-core nodes when compared to pure MPI, although special care is required with NUMA architectures to prevent the reduction of the memory bandwidth of CPUs. This suggest that an optimal usage of the computational resources can be obtained by placing a single MPI process in each NUMA region.

8 Acknowledgements

We acknowledge “Centro per le Tecnologie Didattiche e la Comunicazione” University of Bergamo for the resources provided by CINECA, within the “Convenzione di Ateneo Università degli Studi di Bergamo”, for the availability of high performance computing and support.

REFERENCES

[1] F. Bassi, A. Crivellini, S. Rebay, and M. Savini. Discontinuous Galerkin solution of the Reynolds averaged Navier-Stokes and k - ω turbulence model equations. *Computers & Fluids*, (34):507–540, 2005.

[2] A. Crivellini, V. D’Alessandro, and F. Bassi. A Spalart-Allmaras turbulence model implementation in a discontinuous Galerkin solver for incompressible flows. *Journal of Computational Physics*, 241:388 – 415, 2013.

[3] A. Crivellini, V. D’Alessandro, and F. Bassi. High-order discontinuous Galerkin solutions of three-dimensional incompressible RANS equations. *Computers & Fluids*, 81:122 – 133, 2013.

[4] A. Crivellini, V. D’Alessandro, and F. Bassi. Assessment of a high-order discontinuous Galerkin method for incompressible three–dimensional Navier–Stokes equations: Benchmark results for the flow past a sphere up to Re=500. *Computers & Fluids*, 86:442 – 458, 2013.

[5] F. Bassi, L. Botti, A. Colombo, A. Crivellini, A. Ghidoni, and F. Massa. On the development of an implicit high-order Discontinuous Galerkin method for DNS and implicit LES of turbulent flows. *European Journal of Mechanics, B/Fluids*, 2015. article in press.

[6] C. C. de Wiart and K. Hillewaert. Development and validation of a massively parallel high-order solver for DNS and LES of industrial flows. In *IDIHOM: Industrialization of High-Order Methods-A Top-Down Approach*, pages 251–292. Springer, 2015.

[7] Martin J. Chorley and David W. Walker. Performance analysis of a hybrid MPI/OpenMP application on multi-core clusters. *Journal of Computational Science*, 1(3):168 – 174, 2010.

[8] F. Brezzi, G. Manzini, D. Marini, P. Pietra, and A. Russo. Discontinuous Galerkin approximations for elliptic problems. *Numer. Methods Partial Differential Equations*, 16:365–378, 2000.

[9] D. Hänel, R. Schwane, and G. Seider. On the accuracy of upwind schemes for the solution of the Navier–Stokes equations. AIAA Paper 87-1105 CP, AIAA, July 1987. Proceedings of the AIAA 8th Computational Fluid Dynamics Conference.

[10] F. Bassi, S. Rebay, G. Mariotti, S. Pedinotti, and M. Savini. A high-order accurate discontinuous finite element method for inviscid and viscous turbomachinery flows. In R. Decuypere and G. Dibelius, editors, *2nd European Conference on Turbomachinery Fluid Dynamics and Thermodynamics*, pages 99–108, Antwerpen, Belgium, March 5–7 1997. Technologisch Instituut.

[11] F. Bassi and S. Rebay. A high order discontinuous Galerkin method for compressible turbulent flows. In C.-W. Shu B. Cockburn, G.E. Karniadakis, editor, *Discontinuous Galerkin Methods. Theory, Computation and Applications*, volume 11 of *Lecture Notes in Computational Science and Engeneering*. Springer-Verlag, 2000. *First Internation Symposium on Discontinuous Galerkin Methods*, May 24–26, 1999, Newport, RI, USA.

[12] D. N. Arnold, F. Brezzi, B. Cockburn, and D. Marini. Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM J. Numer. Anal.*, 39(5):1749–1779, 2002.

[13] F. Bassi and S. Rebay. High-order accurate discontinuous finite element solution of the 2D Euler equations. *J. Comput. Phys.*, 138:251–285, 1997.

[14] Sigal Gottlieb, Chi-Wang Shu, and Eitan Tadmor. Strong stability-preserving high-order time discretization methods. *SIAM review*, 43(1):89–112, 2001.

[15] Yohei Sato, Takanori Hino, and Kunihide Ohashi. Parallelization of an unstructured Navier–Stokes solver using a multi-color ordering method for OpenMP. *Computers & Fluids*, 88:496 – 509, 2013.

[16] Dimitri Komatitsch, David Michéa, and Gordon Erlebacher. Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA. *Journal of Parallel and Distributed Computing*, 69(5):451 – 460, 2009.

[17] Gregor J Gassner and Andrea D Beck. On the accuracy of high-order discretizations for underresolved turbulence simulations. *Theoretical and Computational Fluid Dynamics*, 27(3-4):221–237, 2013.

[18] Marc E Brachet, Daniel I Meiron, Steven A Orszag, BG Nickel, Rudolf H Morf, and Uriel Frisch. Small-scale structure of the Taylor–Green vortex. *Journal of Fluid Mechanics*, 130:411–452, 1983.

[19] JJ Gottlieb and CPT Groth. Assessment of riemann solvers for unsteady one-dimensional inviscid flows of perfect gases. *Journal of Computational Physics*, 78(2):437–458, 1988.

[20] Wim M Van Rees, Anthony Leonard, DI Pullin, and Petros Koumoutsakos. A comparison of vortex and pseudo-spectral methods for the simulation of periodic vortical flows at high reynolds numbers. *Journal of Computational Physics*, 230(8):2794–2805, 2011.

[21] Z.J. Wang, Krzysztof Fidkowski, Rémi Abgrall, Francesco Bassi, Doru Caraeni, Andrew Cary, Herman Deconinck, Ralf Hartmann, Koen Hillewaert, H.T. Huynh, Norbert Kroll, Georg May, Per-Olof Persson, Bram van Leer, and Miguel Visbal. High-order CFD methods: current status and perspective. *International Journal for Numerical Methods in Fluids*, 72(8):811–845, 2013.

[22] G. Karypis and V. Kumar. METIS, a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Technical Report Version 4.0, University of Minnesota, Department of Computer Science/Army HPC Research Center, 1998.

[23] Advanced Micro Devices, Inc. AMD Opteron 6200 series processors, Linux tuning guide, 2012. Downloadable from `http://developer.amd.com/resources/documentation-articles/developer-guides-manuals/`.

[24] Advanced Micro Devices, Inc. AMD Opteron 4200/6200 series processors compiler options quick reference guide, 2012. Downloadable from `http://developer.amd.com/resources/documentation-articles/developer-guides-manuals/`.

[25] Xiaohu Guo, Michael Lange, Gerard Gorman, Lawrence Mitchell, and Michèle Weiland. Developing a scalable hybrid MPI/OpenMP unstructured finite element model. *Computers & Fluids*, 110:227 – 234, 2015. ParCFD 2013.

[26] C.K.W. Tam and J.C. Hardin. *Second Computational Aeroacoustics (CAA): Workshop on Benchmark Problems*. NASA conference publication. NASA, 1997.

[27] P.J. Morris. Scattering of Sound by a Sphere: Category 1: Problems 3 and 4. In C.K.W Tam and J.C. Hardin, editors, *Second Computational Aeroacoustics (CAA) Workshop on Benchmark Problems*, 1997. NASA CP 3352.

[28] A. Crivellini and F. Bassi. A three–dimensional parallel discontinuous Galerkin solver for acoustic propagation studies. *International Journal of Aeroacoustics*, 2(2):157–173, 2003.