JAGUAR: a New CFD Code Dedicated to Massively Parallel High-Order LES Computations on Complex Geometry

Adrien Cassagne, Jean-François Boussuge, Nadège Villedieu, Guillaume Puigt, Isabelle d'Ast, Aurelien Genot

La Société Savante de l'Aéronautique et de l'Espace

50h 3AF International Conference on Applied Aerodynamics 29-30 March – 01 April 2015, Toulouse - France

FP25-2015-puigt

JAGUAR: A NEW CFD CODE DEDICATED TO MASSIVELY PARALLEL HIGH-ORDER LES COMPUTATIONS ON COMPLEX GEOMETRY

50th 3AF INTERNATIONAL CONFERENCE ON APPLIED AERODYNAMICS
Toulouse, France, 29-30 March - 01 April 2015
A. Cassagne [1,2], J-F Boussuge [3], N. Villedieu [4], G. Puigt [5], I. D'Ast[6], A. Genot [7]

[1] Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique, CSG team, 42 avenue Gaspard Coriolis, 31057 Toulouse Cedex (France), Email: cassagne@cerfacs.fr
[2] Centre Informatique National de l'Enseignement Supérieur, 950 rue de Saint – Priest, 34000 Montpellier (France), Email: cassagne@cines.fr
[3] Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique, CFD team, 42 avenue Gaspard Coriolis – 31057 Toulouse Cedex (France), Email: boussuge@cerfacs.fr
[4] Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique, CFD team, 42 avenue Gaspard Coriolis – 31057 Toulouse Cedex (France), Email: villedie@cerfacs.fr
[5] Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique, CFD team, 42 avenue Gaspard Coriolis – 31057 Toulouse Cedex (France), Email: puigt@cerfacs.fr
[6] Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique, CSG team, 42 avenue Gaspard Coriolis – 31057 Toulouse Cedex (France), Email: dast@cerfacs.fr
[7] Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique, CFD team, 42 avenue Gaspard Coriolis – 31057 Toulouse Cedex (France), Email: genot@cerfacs.fr

ABSTRACT

LES of industrial flows is associated with geometrical complexity and requires high order schemes to minimize dissipation and dispersion. To tackle these two issues it is necessary to use unstructured grids and High Performance Computing algorithms. In this context, CERFACS initiated two years ago the development of a new CFD code called ***JAGUAR*** based on a mathematical framework leading to high-level capability for LES. In this paper, many topics for HPC are introduced and solved in order to obtain the best code performance.

1. INTRODUCTION

Some problems encountered by industry cannot be addressed easily by standard approaches based on (Unsteady) Reynolds Averaged Navier-Stokes (URANS) modelling. These problems generally encountered at off-design operating point require to account accurately for turbulence effects. For such configurations, the switch to a Large Eddy Simulation (LES) and sisters (DES, Wall Modelled LES) is nowadays recognized as a viable alternative.

Industrial geometry is by nature complex and in general, geometry cleaning is mandatory to perform URANS simulations at a reasonable cost. Here, the key point is to be able to deal with any geometry of high complexity, with a reduced effort on the CAD cleaning. It is therefore mandatory to consider an unstructured framework in order to build a mesh automatically for any kind of configuration.

LES modelling has shown recently its great power for off-design configurations due to the increase in computational power (supercomputers) over the last decades. In fact, it is well known that LES is generally based on explicit time integration and needs very refined meshes. High Performance Computing (HPC) is therefore the bottleneck in order to reduce both the restitution time and the size of the mesh that a computational core has to account for.

In our opinion, the last key point for a LES solver concerns the choice of numerical schemes. Even if LES can be performed with a standard second-order scheme, the number of degrees of freedom can become intractable. For the sake of clarity, a spectral analysis following the von Neumann approach shows that a second order scheme requires 32 more mesh nodes than a six-order scheme for a comparable dissipation / dispersion behaviour. In our opinion, a high order scheme is desirable to minimize the mesh size.

Finally, a good LES solver has to account for complex geometry with an unstructured grid paradigm and it must deal with high parallel performance in order to reduce the restitution time or to solve the problem by splitting the mesh.

In this paper, we present a new LES solver called ***JAGUAR***. ***JAGUAR*** has been designed to solve all these issues linked with LES. In ***JAGUAR***, Navier-Stokes equations are solved following a Spectral Difference (SD) approach. The SD method is a member of spectral discontinuous techniques and the key points are a polynomial reconstruction of variables inside any mesh element and a Riemann solver to take into account explicitly discontinuity at mesh interface. Another point of importance concerns the HPC capability: the SD method needs a local stencil, avoiding the reconstruction of large stencils using cell-neighbouring algorithms. As any discontinuous spectral approach, the SD algorithm induces many computations on the local stencil. Moreover, the parallel data exchange is quite limited.

In section 1, the basic aspects of the SD method are introduced. The following sections are dedicated the analysis of the performance of several parallelism paradigms: single CPU, single GPGPU, MPI parallelism, GPGPU parallelism using MPI and hybrid OpenMP/MPI approaches. Section 8 presents the conclusion of our efforts.

2. THE SPECTRAL DIFFERENCE (SD) METHOD

Kopriva introduced the SD method in 1998 [1] for structured grids and Liu *et al.* extended it to unstructured grids in 2006 [2]. The principle of the SD is to solve Euler or Navier-Stokes equations in their strong form, avoiding any integral formalism. This point justifies the naming of spectral difference: spectral is linked to polynomial shape functions and difference refers to the finite difference approach. For the sake of clarity, explanations regarding the SD method are dedicated to Euler equations in 1D but the approach is easily extended 2D or 3D flows [3] and to viscous terms using a pure centred formulation [4].

The principle of the SD approach is to assume that the vector of unknowns Q varies as a polynomial with a predefined degree inside each segment. In the following and in order to simplify explanations, we assume that the number of unknowns Q is *p+1*: *p+1* solution points are defined inside the segment, where the solution is known. From those *p+1* values, a *p*-th order interpolation polynomial is built to represent quantity variations inside each mesh segment. Now, let us introduce a pure hyperbolic equation with F the flux function which is considered here as a nonlinear function depending on Q.

As for a Finite Difference approach, solutions are computed in all *i* solution points:

$$\frac{\partial Q_i}{\partial t} + \left(\frac{\partial F(Q)}{\partial x}\right)_i = 0 \qquad (1)$$

Following an explicit time marching process, the solution evolution is known once the derivative of the flux density $F(Q)$ is computed at each solution point. For consistence, the flux density must be a polynomial of degree *p+1*, based on *p+2* values computed at some specific locations called flux points: its derivative is therefore a *p*-th order polynomial. Two kinds of flux points are introduced. Any intern flux point is located between two adjacent solution points. The last two flux points are segment ends. The approach is therefore staggered. The flux at intern flux points is build from nonlinear relations using the extrapolated solution in the flux point. For the two end points, two solutions are extrapolated and the flux is computed using a (approximated) Riemann solver. Finally, the flux polynomial is recovered by interpolation from the flux at flux points and it is differentiated in the solution points. The different algorithm steps are summarized in Figs. 1-6.

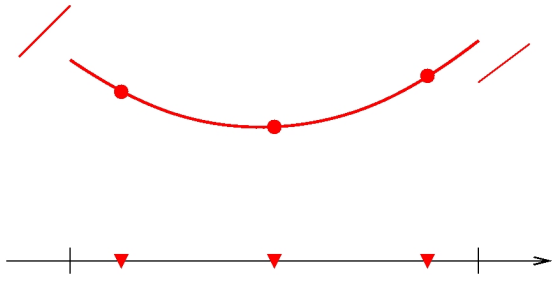

Figure 1: Step 0: Solution points, second order polynomial and solution on the segment.

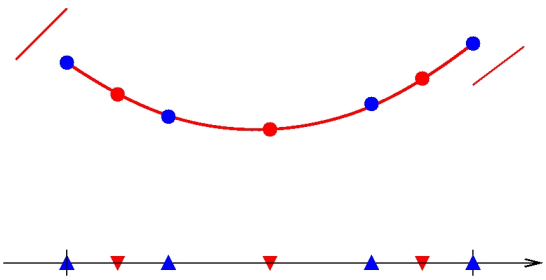

Figure 2: Step 1: Extrapolation of the quantities at flux points.

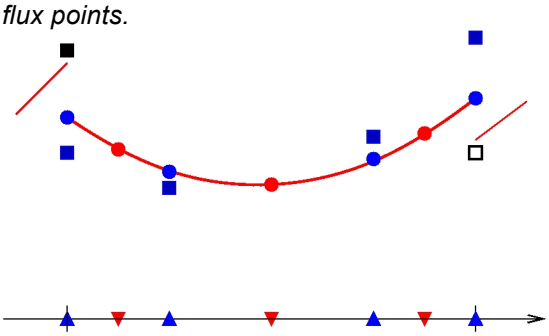

Figure 3: Step 2: computation of the flux from the solution at flux points. Without treatment, two different fluxes are found at segment end points.

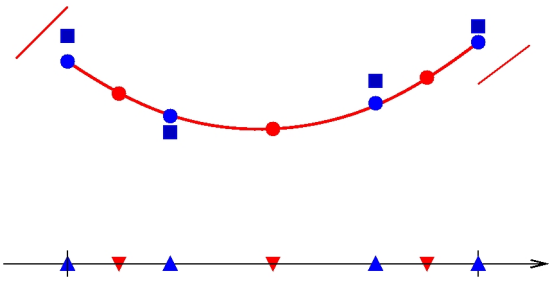

Figure 4: Step 3: unique flux computation at segment end points with a Riemann solver

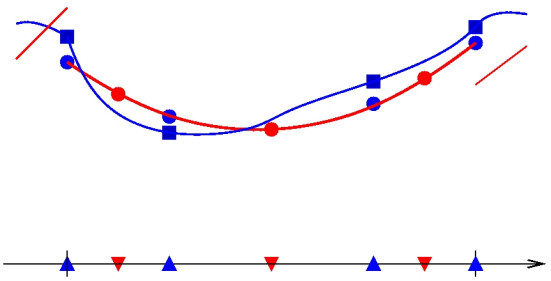

Figure 5: Step 4: New flux polynomial from flux points (at degree p + 1).

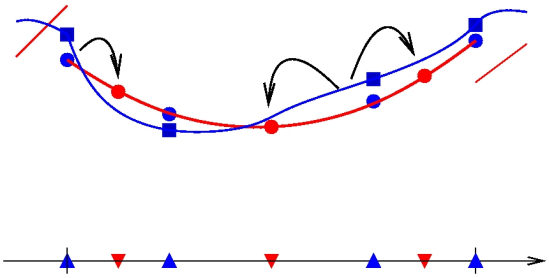

Figure 6: Step 5: differentiation of the flux polynomial at solution points.

The solution and flux polynomials are Lagrange polynomials and the solution points are the following Gauss points on [0,1], as in many papers published in the literature:

$$X_s = \frac{1}{2}\left[1 - \cos\left(\frac{2s-1}{2N}\pi\right)\right] \quad (2)$$

K. Van Den Abeele [5] shows that the definition of the solution point has no effect on both accuracy and stability. The flux point position (defined by index *s+1/2* for a flux point located between solution point *s* and *s+1*) has a strong importance on the numerical stability. In the first papers, the flux points were the Gauss-Lobatto points on [0,1]:

$$X_{s+1/2} = \frac{1}{2}\left[1 - \cos\left(\frac{s}{N}\pi\right)\right], \quad (3)$$

but the scheme is then unstable for an order larger than 2 [5]. In our implementation, we consider the Legendre roots and end points as flux points, for which the SD method is stable [5,6].

The extension to 2D and 3D flows follows a tensor product framework and the SD method can be applied to any unstructured grid composed of hexahedra.

Finally, for memory optimization, the location of unknown is not stored in the physical domain: an isoparametric transformation converts any cell from the physical domain into a regular square element (Fig. 7). Equations are then solved in the isoparametric element and solution is transferred back to the physical domain only when it is mandatory.

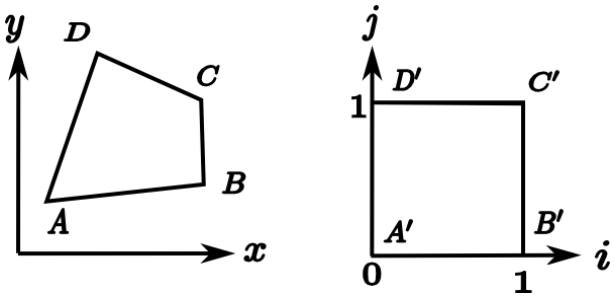

Figure 7: Isoparametric transformation to cast a mesh cell ABCD into the square A'B'C'D'

3. FROM THEORY TO ALGORITHM

Finally, the method can be divided into the following steps:
1. Extrapolation of the solution at the flux points using the *p-th* order polynomial built with solution values Q at the solution points,
2. Computation of the flux in the intern flux points,
3. Treatment of the discontinuity using a Riemann solver to compute the flux on cells faces,
4. Extrapolation of the flux at the solution points using the *p+1-th* order polynomial interpolation F built with computed flux values F at the flux points,
5. Computation of the divergence of the flux polynomial in the solution,
6. Update of the solution Q at the next time step using the flux divergence and a time integration algorithm.

It is then clear that two different treatments must be performed: one inside the mesh element and one dedicated to the flux computation on the mesh faces using a Riemann solver. In the following, the transfer of extrapolated conservative fields at boundary-element flux points into a list of face elements is called the scatter operation. These data are mandatory for the flux computation by the Riemann solver in order to introduce a perfect memory alignment. The transfer of the list of fluxes from face to cells is called the gather operation.

4. SERIAL ALGORITHM OPTIMIZATION

The measured CPU time spent in each routine lead to the following remarks:

1. The computation of the extrapolated solution at flux points and the definition of intern flux (by non linear composition on Q) takes about 40% of the time per iteration.
2. The computation of the flux divergence from the flux polynomial takes about 25% of the time per iteration.
3. The alignment in memory of conservative quantities at end-element flux points (scatter or cell-to-face operation) and the transfer of the Riemann flux to the cells (gather or face-to-cell operation) takes 10% of the time per iteration.
4. The nonlinear Riemann solver takes approximately 10% of the time per iteration.

4.1. Optimization of extrapolation / divergence operations

By nature, a computer is designed to perform matrix / matrix multiplications and compilers are optimized to perform linear algebra. Extrapolation and derivation were written easily in term of matrix / matrix product and memory arrangement has been optimized for vectorization. Moreover, we did not consider here the Basic Linear Algebra Subprogram (BLAS) library because the matrices encountered have a very reduced shape for dedicated libraries like BLAS. Finally, we defined and implemented our own matrix / matrix product.

Moreover, the matrix / matrix product becomes efficient if and only if the memory placement of data is optimized. Our initial memory placement followed a typical placement for a finite volume solver: all conservative variables in a cell are located continuously. Such a memory placement is not optimal for ***JAGUAR*** since extrapolation and derivations are performed variable per variable. Our optimized memory placement has also been considered for flux at flux points.

4.2. Cache blocking

JAGUAR has been designed to deal with *hp*-refinements: in the near future, ***JAGUAR*** will be able to automatically refine the mesh (*h*-refinement) or to increase the polynomial degree (*p*-refinement). A greedy algorithm has been implemented in order to factorize common treatments and reduce the total amount of operations. It consists in making tasks by packing contiguous cells with the same polynomial degree *p*. The greedy algorithm also splits the packed cells into subtasks. These subtasks fit perfectly into the L3 cache of the CPU: it is the well-known cache blocking technique.

4.3. GPGPU implementation

The previous greedy algorithm presents interesting features for General-purpose Processing on Graphics Processing Units (GPGPU), excluding the sub-task creation part. Actually, the greedy algorithm leads to a SIMD (Single Instruction Multiple Data) paradigm inside a given task. This is close to the actual SIMT (Single Instruction

Multiple Threads) paradigm for GPGPU and adaptation to SIMT was obvious.

GPU efficiency mainly depends on the thread bloc configuration regarding memory access of each thread. Here, the goal is to create the right number of threads in order to perform coalesced access in the global memory. For ***JAGUAR***, there are as many GPU threads created as the number of solution points is and each GPU bloc is dedicated to several cells. ***JAGUAR*** uses a fine-grained computing approach.

5. MPI OPTIMIZATION

The MPI library enables parallel computations for both the CPU and GPGPU versions of ***JAGUAR***. For efficiency, the MPI communications are overlapped by computations. Moreover, the chosen persistent communications avoid redundancy in setting up the message each time it is sent.

6. OpenMP IMPLEMENTATION

A simple but not efficient way to introduce OpenMP inside a CFD code consists in creating / pooling OpenMP threads in each routine of the time loop. Our first multi-threading implementation using OpenMP consisted in building a parallel zone (thread pooling zone) for any subroutine inside the time loop. It is a way to get an OpenMP code without many modifications of the original and sequential CFD code. Such a solution presents some drawbacks: there were many unnecessary parallel zone creations and many unnecessary implicit thread synchronisations. For a small number of threads (< 6), the initial version of the code showed relatively good speed-up but for a larger number of threads, the scalability of the code did not meet our expectations. The alternative approach consists in calling the OMP threads creation only one time, just before the time loop in the code. The subroutines dedicated to MPI communications are intrinsically sequential and “OpenMP master zones” was used for them. Except for these subroutines, the whole solver (code inside the time loop) has been entirely parallelized.

In ***JAGUAR***, a heterogeneous OMP strategy has been considered. It practice, some parts of the code are perfectly regular and it is easy to use a standard OpenMP for-loop indices distribution. In contrary, other parts of the code work on mesh cells and the code needs to account for different number of degrees of Freedom (directly depending on the polynomial approximation used for the current cell). As a consequence, it is not possible to perform a for-loop indices distribution and in these cases, the OMP tasking model is applied. In order to reduce the cost of task creation, it is mandatory to be able to allocate more than one cell to a single task. The number of cells associated to one task depends on the size of the cell representation in memory. Actually, the size of the different types of cells has been pre-computed in order to be capable to create cell packs smaller than the L3 cache size. This way the code performs cache blocking and the number of tasks is drastically reduced (by limiting tasks management overhead). We also merged some subroutines in order to improve data independence between threads inside a task. This improvement reduces the number of synchronisations and increases the cache blocking effect. We also deleted some barriers between the subroutines in the solver by using the “no wait” clause wherever possible.

To summarize, our approach consists in calling the OpenMP threads creation only once, just before the time loop in the code. The subroutines dedicated to MPI communications are intrinsically sequential and a serial treatment (“OpenMP master zones”) was used for them. Except for these subroutines, the whole solver has been entirely multi-threaded with OpenMP. In order to reduce the cost of task creation, ***JAGUAR*** relies on a greedy algorithm able to affect many cells to a single task. The number of cells associated to one task depends on the size of the cell representation in memory: the goal is to perform cache blocking. The multi-threading implementation in JAGUAR is fully described in [14].

7. RESULTS AND DISCUSSION

In the following, only the efficiency of Euler computations is introduced. We are currently extending MPI, OpenMP and CUDA strategy to the diffusion term of the Navier-Stokes equations and our first (preliminary) results show that the same kind of conclusion can be drawn for the Navier-Stokes version of ***JAGUAR***. Non integer numbers are defined as double-precision floating-point numbers.

7.1. Serial implementation

The first analysis concerns the serial efficiency of our solver ***JAGUAR***. To do so, we consider a Cartesian grid composed of 128x128 mesh elements. An isentropic vortex, solution of the Euler equations, is convected in a periodic domain. This simulation is inspired by the one of the

International Workshop on High Order CFD methods [7,8]. Here, ***JAGUAR*** V1 will refer to the initial version of the CFD code, without a dedicated optimization, while ***JAGUAR*** V2 is the version with all optimizations presented in Secs. 4.1 and 4.2. Fig. 8 shows that the CPU performance has been highly increased and the restitution time for the same simulation has been decreased by 30% for p=6 (7^{th} order accurate solution).

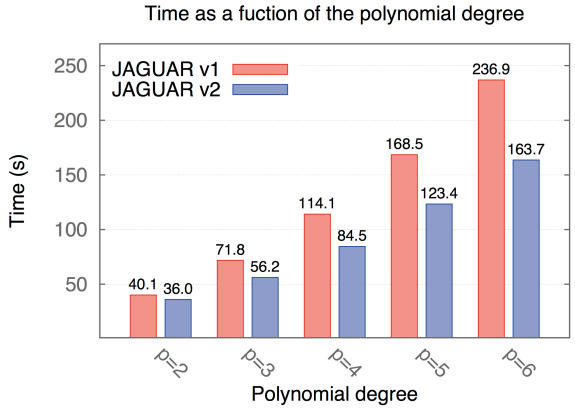

*Figure 8: Computational time (in s) as a function of the polynomial degree for both initial and optimized versions of **JAGUAR**.*

Moreover, the GigaFlops per second (GFlops) of ***JAGUAR*** as a function of the polynomial degree of the solution are shown in Fig. 9. For both versions of the solver, the amount of scalar (referred as scalar) and vectorized (referred as vector) operations are represented. The GFlops per second of the vectorized routines are highly increased and for any order of accuracy, ***JAGUAR*** takes more than 1 GFlops/s. It must be noted that the increase in GFlops/s varies between 27% and 55%.

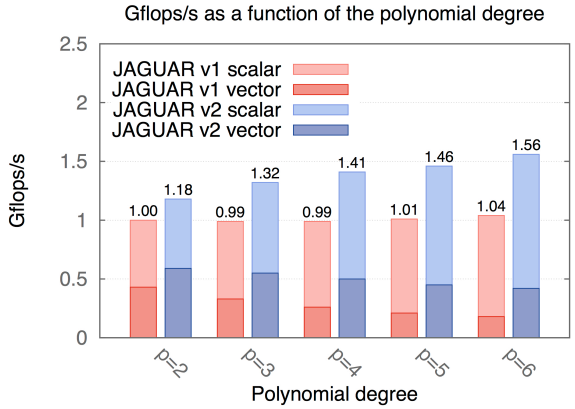

Figure 9: Number of GFlops per second as a function of the polynomial degree of the solution variables.

These computations have been performed on an Intel Xeon E5-2670 CPU that delivers a peak power of 83.2 GFlops/s for 8 cores, and therefore 10.4 GFlops/s for 1 core. ***JAGUAR*** can take benefit of 15% of the peak power and this results makes ***JAGUAR*** one of the most efficient CFD codes available in CERFACS.

7.2. GPGPU implementation on one GPU

In this section, the same kind of results as for the serial implementation is presented for a computation on a 2D 512x512 and a 3D 30x30x30 mesh elements on one Nvidia Tesla K20c GPU. Here, the reference is the optimized version of ***JAGUAR*** called ***JAGUAR*** V2. The mean speed-up is about 30 for both meshes and for any value of p (Fig. 10). The number of floating operations per second is quite low compared to the peak power of the GPU (Fig. 11): 1 170 GFlops/s. But this result is in agreement with other results from the literature [9]. Our experience shows also that the GPU performance strongly depends on the quantity of work to perform on the GPU: maximum efficiency is possible when many operations can be performed and therefore when the code is compute-bound. Unfortunately JAGUAR is memory-bound.

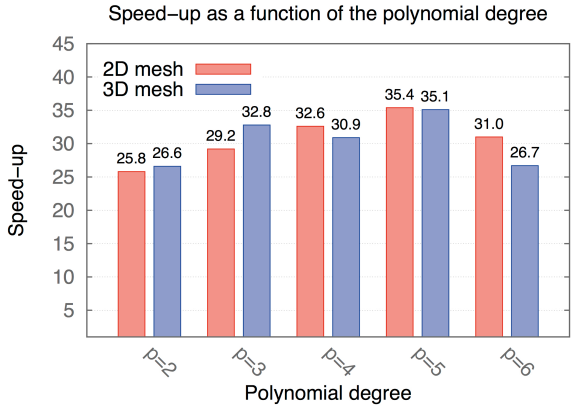

Figure 10: Speed-up obtained on the GPU for 2D and 3D meshes as a function of the polynomial degree p.

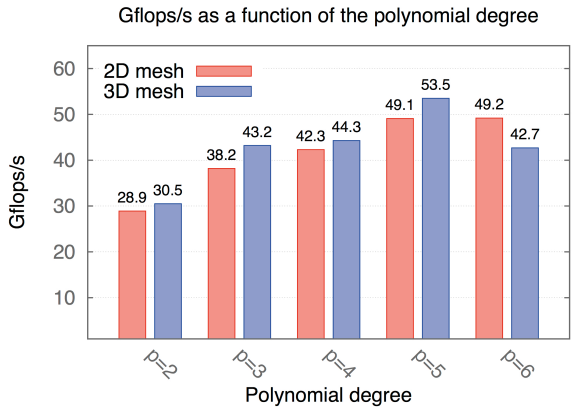

Figure 11: GFlops obtained on the GPGPU for 2D and 3D meshes as a function of the polynomial degree p.

7.3. MPI implementation

The MPI implantation of ***JAGUAR*** has been analysed on Airain platform (Bull supercomputer at CCRT). The mesh considered is composed of 512 x 512 2D elements and a strong scaling analysis is performed using 16, 32, 64, 128, 256, 512, 1024 and 2048 cores. An almost linear speed-up has been obtained for a polynomial degree from 3 to 6 (Fig. 12).

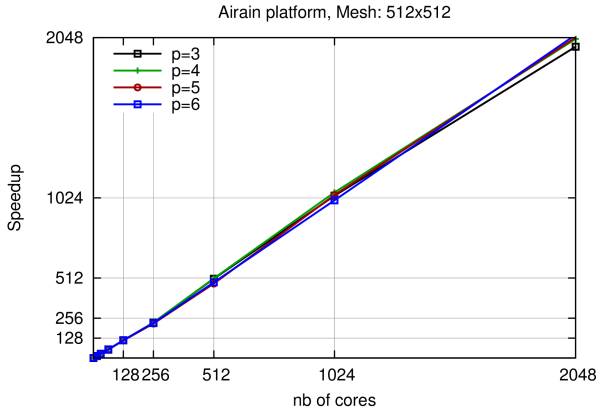

Figure 12: Speed-up depending on the number of computational cores for different order of polynomial reconstruction.

The floating-point operations per second have also been measured on the same mesh and they are summarized in Fig. 13. Two conclusions are drawn. First, when the polynomial degree increases, the work to perform on each mesh cell increases and due to optimizations, the GFlops follow the same tendency. Moreover, GFlops obtained are almost twice as big as standard CFD solvers: the Spectral Difference approach is a good candidate to take benefit from the core performance.

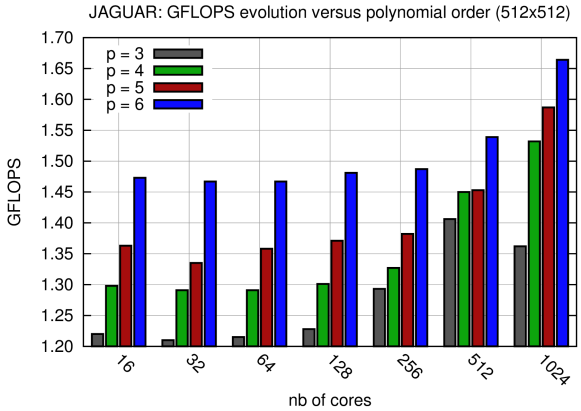

Figure 13: GFlops for several order of accuracy as a function of the number of cores.

Moreover, the time per iteration and per degree of freedom is represented in Fig. 14. For any order of accuracy, an averaged mean time of 2.2 microseconds per iteration and per degree of freedom is measured. It can be deduced that the mean time per iteration is lower than the one for industrial CFD solvers. Moreover, the higher efficiency for 1024 cores seems to be a consequence of the lower size of data per core, leading to a better memory management (cache effect).

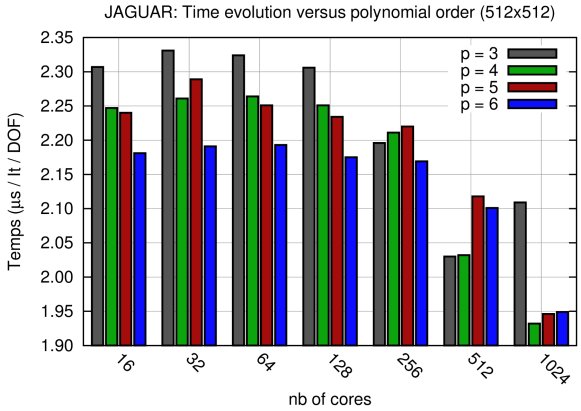

Figure 14: Mean time (in microsecond per iteration and per degree of freedom) depending on the polynomial order and on the number of computational cores.

Our last result concerns an analysis of parallel performance on a less refined mesh. Computations are performed on Neptune [11], one of CERFACS in-house supercomputers based on the same architecture as Airain. Here, a 128x128 regular mesh is split for parallel computations up to 1024 cores (Fig. 15). For such a mesh, the linear scalability is lost: the time spent in communications is large compared to the time spent in

computational loops for this mesh. But ***JAGUAR*** keeps 80% of efficiency on 1024 cores with only 400 degrees of freedom per core. For *p=4* (fifth-order accurate solution), it corresponds to 16 cells in 2D and 3 cells in 3D. It is much lower than for any regular industrial CFD solver and it is also a proof that our implementation of the SD approach is well optimized for parallel simulations.

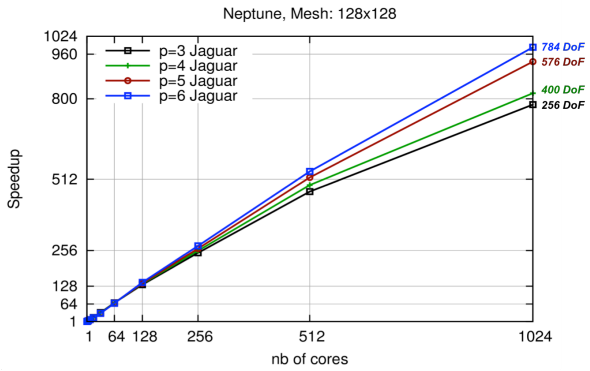

Figure 15: Speed up analysis depending on the number of cores for a small mesh composed of 16,384 quadrangles.

7.4. GPGPU MPI implementation

The parallel version of our GPGPU-enabled solver follows the same implementation optimizations as the regular MPI version. In particular, the code accounts for asynchronous communications and the CPU associated with the GPU performs MPI exchanges. The overall performance is therefore lower than for a pure MPI approach since two data exchanges must be performed: the first one is mandatory to transfer information between the GPU and the CPU while the second one is the MPI exchange between CPUs. On 64 GPU (Fig. 16), the overall efficiency is 47 for a 3D mesh (110 595 hexahedra) and 49.9 for a 2D mesh (640 000 hexahedra). Such a result is in full agreement with results previously published in the literature regarding GPU performance with the same class of discontinuous spectral approaches. Moreover, let's remind that 1 GPGPU has the same power as 4 standard CPU's or 30 cores. For the considered meshes, it means that the configuration is more or less the same as on 30 x 64 = 1920 cores.

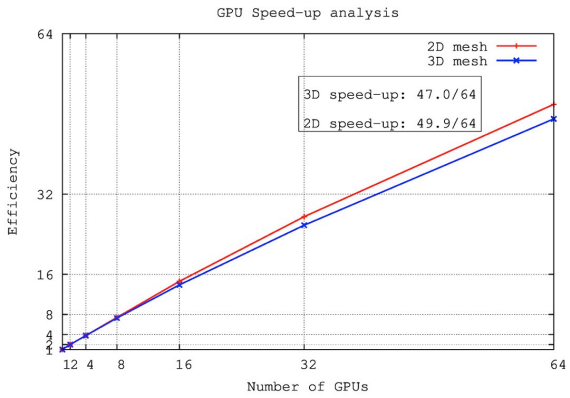

Figure 16: GPU speed-up analysis for both 2D and 3D meshes.

7.5. OpenMP implementation

In the near future, supercomputers may be based on CPU's with many cores and a shared memory. For exascale computing, it could be of strong interest to take benefit of a local parallelism inside the computational node and to decrease both size and number of MPI messages.

The OpenMP computations are performed on the Curie super-computer [10] and we used Curie thin nodes. Any node is composed of two sockets and each socket contains an Intel Sandy Bridge with 8 cores (Hyper-threading was disabled). Our new full-OpenMP implementation leads to an efficiency of 13/16 on the node and a hybrid approach considering MPI between sockets and OpenMP inside socket leads to a speed-up of 14/16 (Fig. 17).

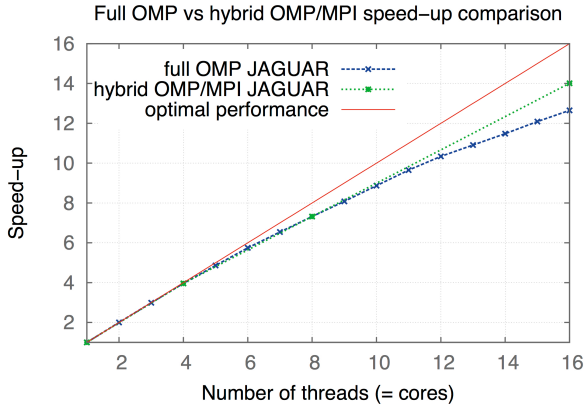

Figure 17: performance obtained with full OpenMP versus hybrid OpenMP MPI approaches.

On the same mesh, a full MPI approach performs slightly better (15-fold speed-up against 14-fold speed-up) as shown in Fig. 18. It is important to note that the MPI parallel efficiency is not optimal for the considered small grid.

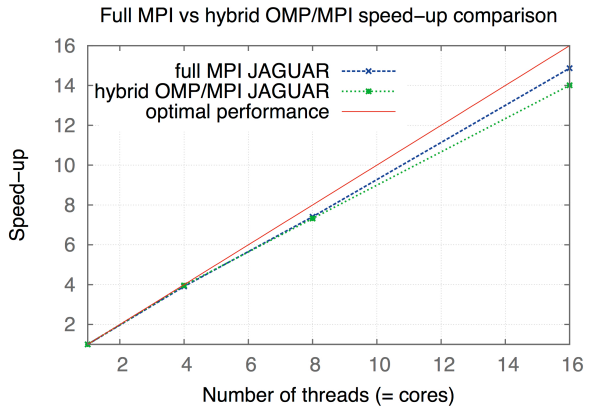

Figure 18: full MPI versus hybrid MPI / OpenMP parallel performance

8. CONCLUSION AND PERSPECTIVES

Large Eddy Simulation is entering now in industry. For this purpose LES implementation needs several prerequisite: unstructured grid capability, high efficiency on a core, parallel efficiency and high order of accuracy. We have presented in this paper a new technique to address these constraints. The class of discontinuous spectral approaches considers a polynomial reconstruction of data inside the element and discontinuous solution at cell interfaces. Beyond the class of discontinuous spectral approaches, we considered the spectral difference approach: equations are solved in their strong differential form, which avoids many complex integral quadrature rules encountered in many other methods of the same class (Discontinuous Galerkin, Spectral Volume approaches).

During the last two years, attention was focused on the serial and parallel efficiency of our solver and we obtained important results. First of all, high order discontinuous spectral approaches are assumed to lead to a large CPU cost. Our experience shows that the CPU cost of our new solver ***JAGUAR*** is lower than the one of standard industrial solvers. This is possible since there are many operations to perform on a reduced stencil and GFlops are almost twice larger than with regular CFD codes. Moreover, these methods are designed to MPI computations by nature and it is quite easy to obtain a linear speed-up. Obtaining a high efficiency in full OpenMP is complicated at the present time since classic nodes contain two sockets. Moreover, the task model is sub-optimal when there are two memory banks: it is possible that a core from a socket wants to access the memory bank of the other socket. The better efficiency on a node is still attained with a pure MPI approach but a hybrid technique based on OpenMP and MPI is now competitive. In the near future, the number of cores on the node will increase and the hybrid approach will be mandatory in order to decrease exchanges managed by the MPI library.

We are now focusing attention on two main topics. The first one concerns the adaptation of ***JAGUAR*** for massively computations considering a Many Integrated Core architecture. The second topic concerns the extension of the solver to handle academic and industrial configurations: boundary conditions, Input/Output, co-treatment using our Antares library [12], *hp*-adaptation. It is also planned to analyse the present capability of ***JAGUAR*** to treat academic combustion flames.

ACKNOWLEDGEMENT

The work presented in this paper was partially supported by Safran. Moreover, some aspects on High Performance Computing were highlighted during a PRACE preparatory access of type C [13] and the authors thank the CINES for its support on the OpenMP implementation optimization.

REFERENCES

[1]. Kopriva,D.A. (1998). A staggered-grid multi-domain spectral method for the compressible Navier-Stokes equations, *Journal of Computational Physics,* **143**, 125-158.

[2]. Liu, Y., Vinokur, M., Wang, Z.J. (2006). Spectral Difference method for unstructured grids I: Basic formulation. *Journal of Computational Physics,* **216**, 780-801.

[3]. Sun, Y.; Wang, Z.J. & Liu, Y. (2007). High-Order Multidomain Spectral difference method for the Navier-Stokes equations on unstructured hexahedral grids. *Communications in Computational Physics,* **2**, 310-333.

[4]. Sun, Y.; Wang, Z.J. and Liu, Y. (2006). High-Order Multidomain Spectral Difference Method for the Navier-Stokes Equations. In *44th AIAA Aerospace Sciences Meeting and Exhibit, AIAA Paper 2006-301.*

[5]. Van den Abeele K., Lacor C. and Wang Z.J. (2008). On the stability and accuracy of the spectral difference method. *Journal of Scientific Computing*, **37**,162-188.

[6]. Jameson A. (2010). A proof of the stability of the spectral difference method for all orders of accuracy. *Journal of Scientific Computing*, **45**, 348-358.

[7]. Second International Workshop on High Order CFD Methods (2013), May 27-28, Cologne.

[8]. Villedieu N., Puigt G. and Boussuge J-F. (2013). High Order Workshop: computations performed with ***JAGUAR***, an in-house CFD code based on the spectral difference formalism, In *2nd International Workshop on High-Order CFD Methods*, May 27-28, Cologne.

[9]. Castonguay P.; Williams, D.M.; Vincent, P.E.; Lopez, M. and Jameson A. (2011). On the development of a high-order, multi-GPU enables, compressible viscous flow solver for mixed unstructured grids. In *20th AIAA Computational Fluid Dynamics Conference*, Honolulu, HawaII, AIAA Paper 2011-3229.

[10]. Airain super-computer web page: *http://www-ccrt.cea.fr/fr/moyen_de_calcul/airain.htm*

[11]. Neptune super-computer description : *http://www.cerfacs.fr/files/cerfacs/computing/BULL.jpg*

[12]. Curie super-computer web page: *http://www-hpc.cea.fr/fr/complexe/tgcc-curie.htm*

[13]. Gomar, A.; Léonard, T. and others (2012-…). *Antares, python post-processing library*. URL: http://www.cerfacs.fr/antares/

[14]. Cassagne, A.; Puigt, G. and Boussuge, J-F. (2015). High-order Method for a New Generation of Large Eddy Simulation Solver (HORSE), *Partnership for Advanced Computing in Europe (PRACE)* Preparatory Access of Type C, Final report.