Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Numerical Simulation of complex turbulent Flows with Discontinuous Galerkin Method

A. Abbà[a*], A. Emerson[b], M. Nini[a], M. Restelli[c], M. Tugnoli[a]

[A]*Dipartimento di Scienze e Tecnologie Aerospaziali, Politecnico di Milano, Via La Masa, 34, 20156 Milano, Italy*
[b]*Cineca, via Magnanelli 6/3, 40033 Casalecchio di Reno, Bologna, Italy*
[c]*Max-Planck-Institut für Plasmaphysik, Boltzmannstraße 2, D-85748 Garching, Germany*

Abstract

We present a performance analysis of the numerical code DG-comp which is based on a Local Discontinuous Galerkin method and designed for the simulation of compressible turbulent flows in complex geometries. In the code several subgrid-scale models for Large Eddy Simulation and a hybrid RANS-LES model are implemented. Within a PRACE Preparatory Access Project, the attention was mainly focused on the following aspects:

1. Testing of the parallel scalability on three different Tier-0 architectures available in PRACE (Fermi, MareNostrum3 and Hornet);
2. Parallel profiling of the application with the Scalasca tool;
3. Optimization of the I/O strategy with the HDF5 library.

Without any code optimizations it was found that the application demonstrated strong parallel scaling of more than 1000 cores on Hornet and MareNostrum, and least up to 16384 cores on Fermi. The performance characteristics giving rise to this behaviour were confirmed with Scalasca. As regards the I/O strategy, a modification to the code was made to allow the use of HDF5-formatted files for output. This enhancement resulted in an increase in performance for most input datasets and a significant decrease in the storage space required. Other data were collected on the influence of the optimal compiler options to employ on the different computer systems and the influence of numerical libraries for the linear algebra computations in the code.

1. Introduction

The Discontinuous Galerkin (DG) finite element method is characterized by high parallelizability and accuracy. These characteristics make the DG approach very suitable for numerical simulations of flows in complex geometries on supercomputers. At the same time, the Large Eddy Simulation (LES) is the most interesting and promising technique for the simulation of turbulence. The attractive feature of coupling the two lies in the possibility of exploiting the hierarchical representation of the unknowns naturally provided by DG formulations at the scale separation lying at the basis of the LES approach.

This white paper describes the analysis of the performances of the high-order accuracy numerical code DG-comp suitable for numerical simulation of turbulent compressible flows on Tier-0 systems. The numerical code is based on the FEMilaro library, a finite element library available under GPL3 license at the link "http://code.google.com/p/femilaro/". The computations have been performed on the IBM BlueGene/Q system FERMI at CINECA (Bologna), on the Cray XC40 HORNET at HLRS (Stuttgart) and on MareNostrum at BSC (Barcelona) which were made available thanks to the award of a PRACE preparatory access project (Type C).

* Corresponding author. *antonella.abba@polimi.it*

2. The numerical code

In the DG-comp numerical code employed for the project the Navier-Stokes equations for an unsteady, compressible flow are solved for the conserved variables of density, momentum and specific total energy. The equations are discretized in space using a Local Discontinuous Galerkin (LDG) method [1][2][3] on tetrahedral elements. With the LDG method, the equations written in weak form are projected in a space of polynomial functions which are defined in each element of the computational grid. In this way the numerical solution is obtained by solving a system of ordinary differential equations on each element, and the coefficient matrix of the system is built locally. The LDG method may then reach a high degree of accuracy using high order polynomial functions, and at the same time presents characteristics of high parallelization. Although the code is written for a generic polynomial degree, for a typical turbulent simulation polynomials of fourth degree are used, resulting in a fifth-order method.

Since the finite element space is discontinuous, there is great freedom in the choice of the local polynomial space for the basis functions. Choosing a modal representation, i.e. such that the local basis functions are orthogonal, leads to a diagonal mass matrix and simplifies the time integration. Moreover, the local polynomial representation directly provides a means to separate "large scale" modes from "small scale" modes, thus providing the starting point for the definition of the turbulence models.

The coupling among the element degrees of freedom is obtained by means of numerical fluxes, as is typical in the context of DG methods [2]. In the present implementation, the Rusanov numerical flux is used for the conserved quantities [3].

In the LDG method, auxiliary variables are introduced to represent the gradients of the primitive variables, velocity and temperature. Such gradients are then used to compute both the diffusive and the turbulent momentum and energy fluxes. It is important to notice that these variables are computed locally from the corresponding primitive variables, and hence they do not result in a major additional computational cost, nor communication overhead. These variables also require some numerical fluxes, for which the classical Bassi-Rebay definition [2] is adopted. For the time integration the fourth-order, strong stability preserving Runge-Kutta scheme (SSPRK) proposed in [4] is used. Nevertheless other time integration schemes are available: the classical explicit Runge-Kutta scheme up to fourth-order, and matrix-free exponential time integrators based either on Krylov space techniques [5]or on Leja point interpolation [6], for example.

Discontinuous finite elements provide a natural framework to generalize LES filters to arbitrary computational meshes. The filter operator that is the key tool in LES can be identified with the projection operator on a finite dimensional space related to the discretization [7]. This allows us to generalize easily the LES concept to unstructured meshes and complex geometries, and a variety of subgrid scale (sgs) models for LES [8] and hybrid RANS/LES models [9] are implemented in the code.

The code is written following the latest FORTRAN 2003 and 2008 standards, taking specifically advantage of the language support for array variables and exploiting the new object oriented features of the language for a flexible implementation of "tasks" like the time integrator and the turbulence models. This results in great flexibility in choosing and combining different options.

The computational domain is partitioned before the computation, for instance using the METIS software. Each partition is then assigned to a processor, and all the communications use MPI. Since in the present implementation the computational cost is roughly the same for all the elements, there is no need for dynamic load balancing.

Most of the computations are local to each element. This means that, although the computational cost for DG methods is typically higher compared to other formulations, DG methods are well adapted to parallel execution.

The MPI communications are associated with the numerical fluxes and two communications are required for each time step: one for the hyperbolic numerical fluxes and one for the viscous ones. These communications have the same topology, where each processor exchanges data with those processors sharing at least one side. No global communications are required. In practice, this rather simple communication pattern is implemented with point-to-point, nonblocking MPI operations.

The code also includes the computation of some diagnostics such as energy and momentum average fluxes and flux fluctuations. These diagnostics require a more complicated communication pattern, since it is often necessary to compute averages across many processors. Here, various MPI operations are used, both point-to-point and collective and the recently introduced non-blocking collective MPI operations, such as MPI_IALLREDUCE.

3. Results

The following analysis has been carried out for the typical turbulent plane channel flow test case at friction Reynolds number equal to180. The present numerical code and the implemented sub-gridscale models have been tested in this well documented condition. Figure 1 and Figure 2 show respectively a sketch of the geometry and a visualization of the velocity, in three orthogonal planes, obtained at Mach number 0.7. The subgrid scales are modelled with the anisotropic eddy viscosity model [8]. For a plane channel at the cited Reynolds numbers, about 18000 elements and the fourth order degree polynomial guarantee that the laminar sublayer and the turbulence structures in the wall region are well reproduced. In the present project, for the scalability test, a grid with 316800 tetrahedral elements has been used. This corresponds to the mean size of more complex LES simulations planned for possible future TIER-0 project.

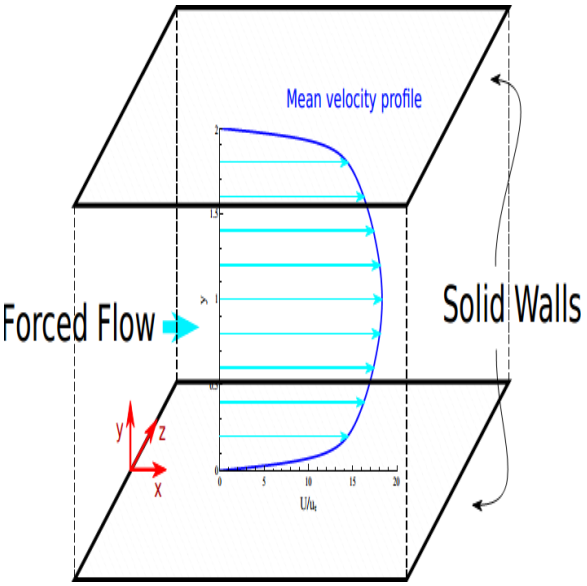

Figure 1. Geometry of turbulent plane channel flow

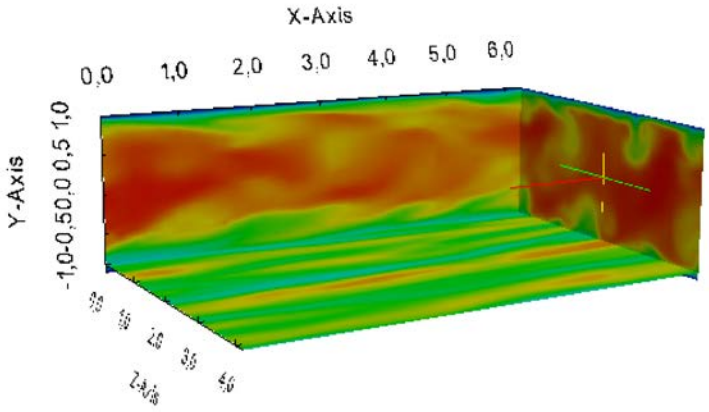

Figure 2. Visualization of the velocity at Mach number 0.7.

3.1 Scalability analysis

A strong scalability analysis has been performed for the turbulent channel flow simulation. The wallclock times have been evaluated for the time advancing computation, neglecting the time for initialization procedures, input and output. The analysis has been carried out with a range of processor cores varying from 1024 to 16384 on FERMI, with 24 up to 1536 cores on HORNET and 16 to 2048 cores on MareNostrum with one MPI rank per

core for each run. Note that in principle higher numbers of cores are available on these platforms but the tests we have performed represent the usual maximum limits used in LES simulations. For each platform, the speedup is normalized by the performance obtained with the lowest number of processor cores and these data are reported in Tables 1-3. In Figures 3-5 the scaling obtained with DG-comp on the three different platforms are compared with the linear optimum trend. The figures confirm the very good scalability properties of the numerical code, as expected from the characteristics of the DG method described above.

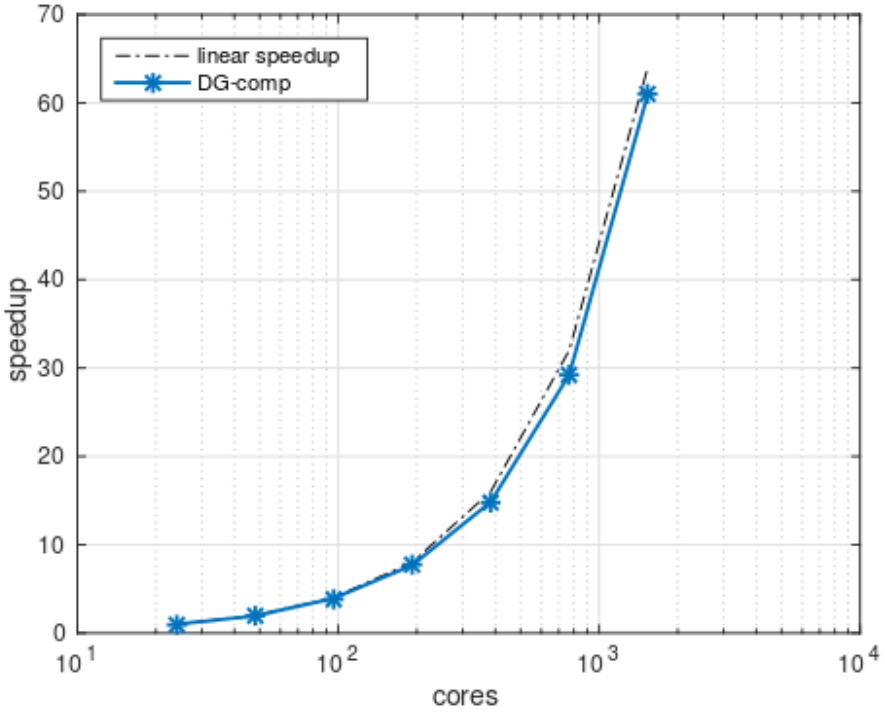

Figure 3 Strong scaling on Hornet

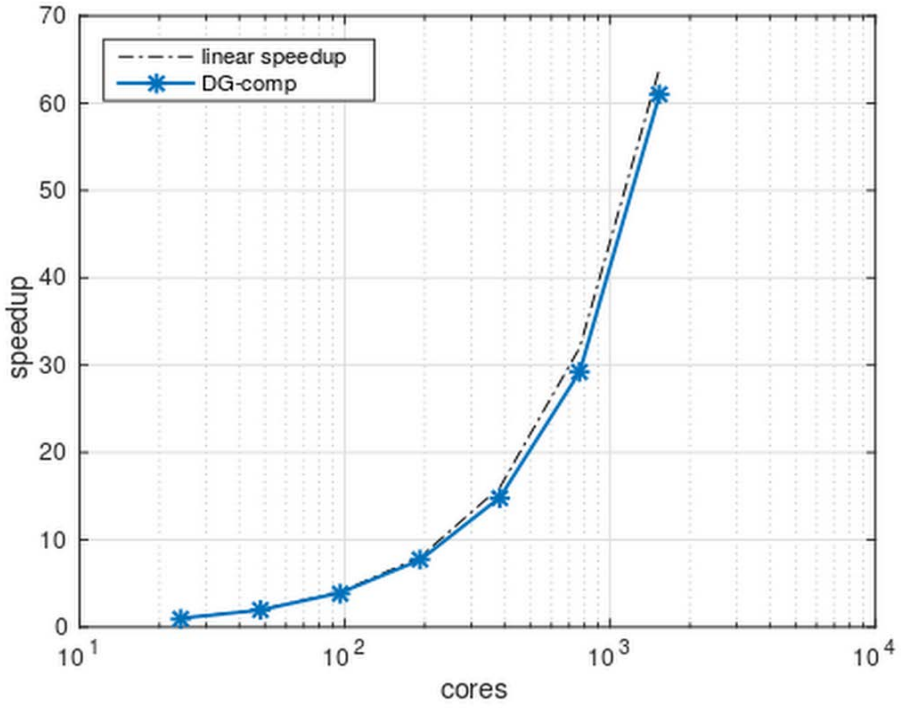

Figure 4 Strong scaling on Mare Nostrum

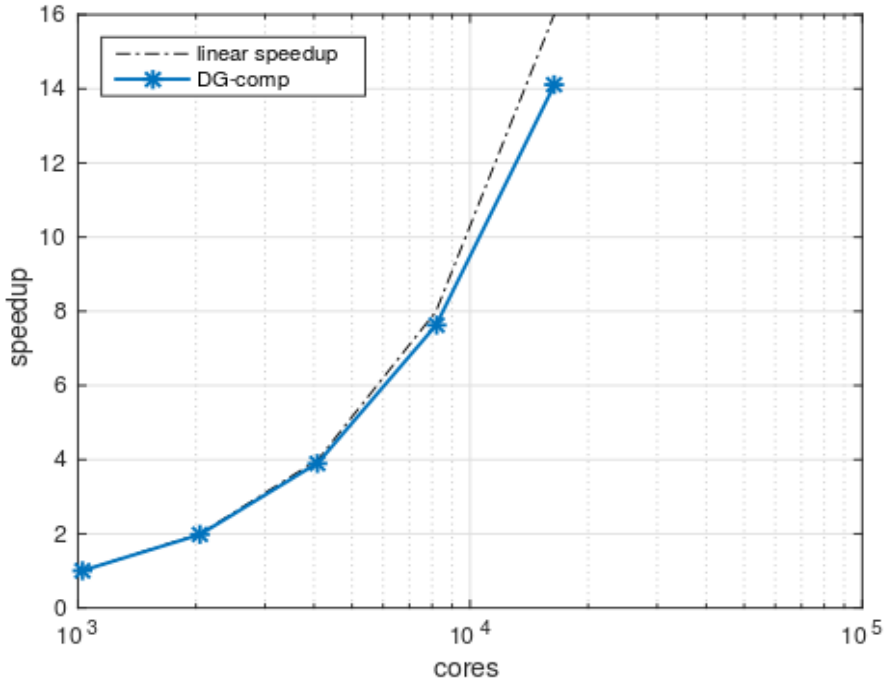

Figure 5 Strong scaling on Fermi

Number of cores	Wall clock time [s]	Speed-up vs the first one	Number of Nodes
1024	8.1383	1	64
2048	4.1183	1.9761	128
4096	2.0793	3.9139	256
8192	1.0675	7.6237	512
16384	0.5776	14.0901	1024

Table 1. Performance data from the simulations on Fermi

Number of cores	Wall clock time [s]	Speed-up vs the first one	Number of Nodes
24	40.2157	1	1
48	20.7937	1.9340	2
96	10.2777	3.9129	4
192	5.2527	7.6562	8
384	2.7128	14.8243	16
768	1.3760	29.2265	32
1536	0.6596	60.9711	64

Table 2. Performance data from the simulations on Hornet

Number of cores	Wall clock time [s]	Speed-up vs the first one	Number of Nodes
16	52.5055	1	1
32	26.4568	1.9846	2
64	13.3011	3.9475	4
128	6.7066	7.8289	8
256	3.3634	15.6109	16
512	1.7149	30.6179	32
1024	0.8907	58.9492	64
2048	0.4802	109.3471	128

Table 3. Performance data from the simulations on Mare Nostrum.

3.2 Compilation options

Some different compiler options have been tested on Mare Nostrum and Hornet. The results are presented in Tables 4-6. On Hornet, due to older versions of the GNU compiler and compatibility issues with the other compilers, only the Intel Fortran compiler has been tested. Instead, on Mare Nostrum both the GNU Fortran compiler and the Intel Fortran compiler have been tested; 72 and 64 cores have been used on HORNET and MareNostrum respectively.

The time of all the preliminary operation has been recorded separately from the time spent in the main time evolution cycle, performed for this test only on 150 iterations. We tested different optimization levels, from -O1 to -O3. The -O0 optimization obviously performed much worse than the others.

We observed that, while the time spent before the main computing cycle decreases with optimization, the time advancing cycle is affected by a decrease of the performances on increasing the optimization level. On Intel compilers the best performances were achieved with -O1, while with Gnu compilers the performance peak was observed with -O2.

Moreover we can observe that on MareNostrum the GNU compiler performs a little better than the Intel compiler.

options	time before cycle	cycle time	total time
-O1 -g -fpe0	46 .0	149 .7	197 .4
-O2 -g -fpe0	41.8	154.9	197.8
-O3 -g -fpe0	41.1	164.8	207.3

Table 4 Intel compiler runs on Hornet

options	time before cycle	cycle time	total time
-O1 -g -fpe0	37.5	138.5	176.6
-O2 -g -fpe0	38.8	145.5	184.8
-O3 -g -fpe0	36.0	154.0	190.4

Table 5. Intel compiler runs on Mare Nostrum.

options	time before cycle	cycle time	total time
-O1 -g –ftrap	14.4	136.5	151.0
-O2 -g –ftrap	12.7	130.7	143.3
-O3 -g –ftrap	13.2	152.7	166.3

Table 6. GNU compiler runs on Mare Nostrum

3.3 Profiling

A profiling analysis of the numerical code has been performed on FERMI using the Scalasca tool and with the code compiled at the –O2 level with the IBM Fortran compiler xlf: 1024 cores for a grid with about 18000

elements have been used. We note that in the following the results limited to the main computation cycle only are presented.

The Scalasca analysis demonstrates that the computational effort is well distributed between the nodes, therefore not justifying a modification to improve the task load balancing scheme. In addition, almost all of the time is consumed in the computation of the right hand side terms of the Navier-Stokes equations (Figure 6). Inside this, the integration over the element volume and the turbulence model are the more time requiring procedures; very little time is spent for diagnostic computations.

We note also that about 18% of the time is spent in communications. The profiling confirms that all the point-to point communications happen in the computation of the hyperbolic numerical fluxes and of the viscous ones, exchanged between elements through element surfaces, while the diagnostics evaluation involves collective MPI operations, such as MPI_IALLREDUCE.

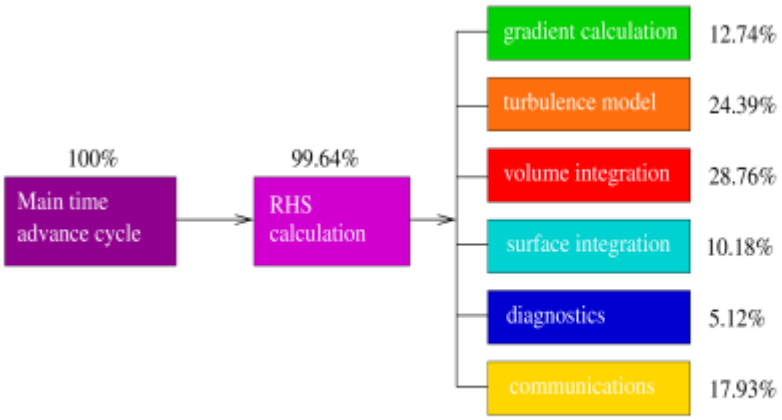

Figure 6 Profiling data obtained from Scalasca

This analysis shows how the computational times appear evenly distributed among the different tasks in the code.

For a better evaluation of the effort spent in communications, profiling tests have been carried out on FERMI for a case with 316800 elements using 1024 and 16384 cores respectively. The results are shown in Figure 7. We can observe that the communication time spent with 18000 elements and 1024 cores is almost the same as the one with 316800 elements and 16384 cores. The reason is due to the fact that the processors perform point-to-point communications only on the partition surfaces with the result that, assuming a good partitioning of the computational grid, the number of messages exchanged by each processor is roughly independent of the number of processors as well as of the problem size.

We have attempted also to examine possibilities for optimizing the numerical calculations. The right hand side calculation in an explicit time stepping DG method consists of a very large number of small calculations, mainly multiplication of matrices of order of magnitude from 10x10 to 100x100. In these calculations no concentrations of computational effort have been discovered, and so no major opportunities of optimization were identified. The used of dedicated numerical libraries, such as the BLAS library, instead of the MATMUL implementation of the Fortran language was tested, but it showed no evident improvement over the computational time spent in the right hand side calculation, most likely due to the relatively small matrices involved in each single multiplication.

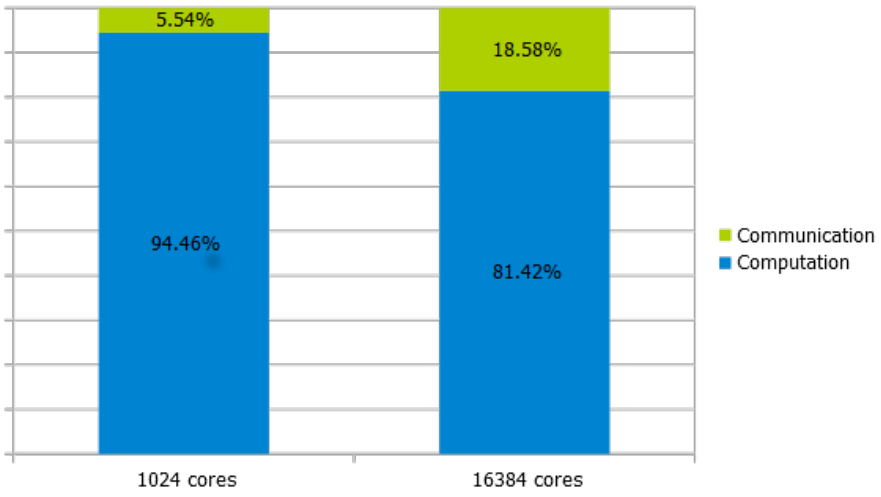

Figure 7. Scalasca analysis showing the relative costs of computation and calculation on Fermi for 1024 and 16384 cores.

3.4 I/O with HDF5

The standard output form of FEMilaro is through a series of ascii formatted files. Usually each processor writes its own portion of the results, as well as grid and base information, statistics and diagnostics, in a file formatted in order to be compatible with Octave, which is currently the main tool used in the postprocessing of the results.

This approach is convenient in the sense that the output procedure is uncomplicated and the output files can be easily examined in ascii format and are ready to be loaded in Octave without the need for any specific parser. The output can also be changed in the number, size and type of variables without the need to change any code in the postprocessing tool chain. Nevertheless in this way the output is inefficient in terms of storage size, output time and number of files generated.

A more efficient solution, enabling parallel I/O without compromising the possibility to have a flexible, easily examinable and loadable output, is available using the parallel HDF5 library.

The major part of the output has been independently ported to the HDF5 standard and now the user can choose to use the HDF5 or the Octave format for either the input and the output. Only a very small part of the output is still written only in the Octave format since it would require the use of variable length arrays written in parallel, a feature which is not currently supported in the current version of the HDF5 library.

The output with HDF5 consists of a single output file written collectively by all the processes at each output time (see Figure 8).

All the output data can be divided into common data and collective data. The common data consist of small variables, usually scalars or three component vectors, which are known equally by all the processes, and are written by a single process. Usually some parameters and statistics describing the simulation status, for example, are common data. The collective data instead consist of the main part of the solution and are composed of large multi dimensional arrays scattered among the various processes which are responsible for the calculation of different parts of these arrays. Instead of keeping these data scattered in the output as in the Octave format, the results are automatically gathered during the write phase and printed in the HDF5 file as a single ordered array. There is no explicit gathering of the data: every process selects its own portion of the array in which to write and then the HDF5 library takes care of writing the data in the correct order collectively in order to obtain the final array.

The additional benefit of this output procedure is that the results are completely independent from the mesh partitioning: the results obtained with some partitioning scheme can be used as the initial condition for another case with the same mesh but different partitioning. This is crucial to enable interoperability of the data between different architectures which require very different numbers of partitions, for example between x86 and BlueGeneQ architectures.

In terms of efficiency the use of HDF5 format instead of the Octave format showed a substantial improvement in terms of storage space, with a similar write time.

The disk usage reduction depends on the problem size and settings, as well as on the number of processes employed. For typical test cases we experienced a reduction ranging between the 63% and the 91% on the different machines.

In Table 7 the mean write time from some test cases on Fermi and Hornet are presented. The write time is decreased on Hornet while it is increased on Fermi, but still remains similar and is considered acceptable since our primary requirement was to collect all the outputs in a single file and to reduce disk usage, given the relatively small number of output times during a typical test case.

It is also necessary to remark that the results shown were obtained with the default HDF5 settings and without any particular optimization or tuning for the different filesystems of the different machines. We expect that the time performance of the HDF5 writing process could be improved with an appropriate tuning of the library, especially for the GPFS filesystem of Fermi.

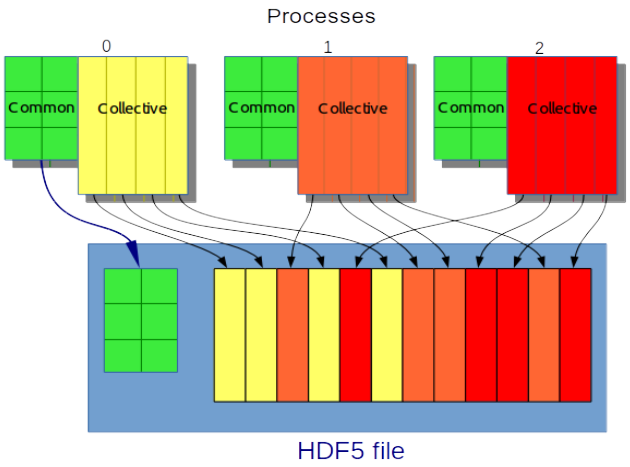

Figure 8. Schematic representation of HDF5 file write procedure.

	Hornet		Fermi	
Output Format	Octave	HDF5	Octave	HDF5
Mean write time [s]	10.81	8.73	4.87	7.39

Table 7. Writing time with and without the use of HDF5 library.

4. Conclusions

In this work we have demonstrated that the DG-comp finite element code for turbulent flows shows a high parallel scalability on three different Tier-0 supercomputers in PRACE, namely Fermi (BG/Q), Hornet (Cray XC40) and Mare Nostrum (Intel cluster). Although, the good scaling behaviour was to be expected from the known characteristics of the code, this project has allowed us to prove that the scaling behaviour is maintained even on very different architectures.

A second major outcome of the project has been the modification of the code to produce output as HDF5 files instead of plain ascii. As well as an increase in performance in most cases, this modification also results in a large reduction in output size. This code enhancement, together with the scaling data and the information obtained on the optimum compiler options, can now be used to construct a strong case for applying for future resource allocations on Tier-0 computers.

References

[1] B. Cockburn, C. W. Shu. The local discontinuous Galerkin method for time-dependent convection-diffusion systems, SIAM J. Num. Anal. 35 (6), 1998.

[2] F. Bassi, S. Rebay. A high order accurate discontinuous finite element method for the numerical solution of the compressible Navier-Stokes equations. J. Comp. Phys., 138 (2), 1997

[3] F.X. Giraldo, M. Restelli. A study of spectral element and discontinuous Galerkin methods for the Navier-Stokes equations in nonhydrostatic mesoscale atmospheric modeling: Equation sets and test cases, J. Comp. Phys. 227 (8), 2008.

[4] S.J. Ruuth, R.J. Spiteri. High-order strong-stability-preserving Runge-Kutta Methods with downwind-biased spatial discretizations. SIAM J. Numer. Anal., 42(3), 974-996, 2004.

[5] J. C. Schulze, P. J. Schmid, J. L. Sesterhenn. Exponential time integration using Krylov subspaces, Int. J. Numer. Meth. Fl., 60 (6), 2009.

[6] M. Caliari, A. Ostermann. Implementation of exponential Rosenbrock-type integrators. Appl. Num. Math., 59, Issues (3-4), 568–581, 2009.

[7] F.van der Bos, J.J.W. van der Vegt, and B.J. Geurts. A multiscale formulation for compressible turbulent flows suitable for general variational discretization techniques. Computer Methods in Applied Mechanics and Engineering, 196, 2863-2875, 2007.

[8] A. Abbà , L. Bonaventura, M. Nini , M. Restelli. Dynamic models for Large Eddy Simulation of compressible flows with a high order DG method. Computers and Fluids, DOI: 10.1016/j.compfluid.2015.08.021, in press

[9] M. Nini, A. Abbà, M. Germano, M. Restelli. Analysis of a Hybrid RANS/LES Model using RANS Reconstruction. Proceeding of iTi2014 - conference on turbulence, Bertinoro, September 21-24, 2014.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-312763 and the EU's Horizon 2020 Research and Innovation Programme (2014-2020) under grant agreement no. EINFRA-653838.