High-order Method for a New Generation of Large Eddy Simulation Solver

Adrien Cassagne, Jean-François Boussuge, Guillaume Puigt

Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

High-order Method for a New Generation of Large Eddy Simulation Solver

A. Cassagne[a*], J-F. Boussuge[b], G. Puigt[b]

[a]*Centre Informatique National de l'Enseignement Supérieur, Montpellier, France*
[b]*Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique, Toulouse, France*

Abstract

We enabled hybrid OpenMP/MPI computations for a new generation of CFD code based on a new high-order method (Spectral Difference method) dedicated to Large Eddy Simulation (LES). The code is written in Fortran 90 with MPI library and OpenMP directives for the parallelization. This white-paper is focused on achieving good performances with the OpenMP shared memory model on standard environment (bi-socket nodes and multi-core x86 processors). The goal was to reduce the number of MPI communications by considering MPI communications between nodes and OpenMP approach for all cores on any node. Three different approaches are compared: full MPI, full OpenMP and hybrid OpenMP/MPI. We observed that hybrid and full MPI computations took nearly the same time for a small number of cores.

Introduction

Industry tends to look for Large Eddy Simulation (LES) in order to address some problems that cannot be captured by classic RANS and URANS techniques. We recall that the principle of LES is to capture the main turbulent structures (vortices) while RANS and URANS technique consider the mean effects of turbulence. When the flow becomes intrinsically unsteady due to flow / turbulence interactions, RANS and URANS models show generally a poor accuracy and capturing turbulent vortices helps to analyse the problem. This switch to LES has many consequences, especially on accuracy and efficiency. LES is associated with the transport of turbulent vortices on large distance and the first question to address is in relation with the degree of accuracy. Low order schemes are more dissipative (loss of wave amplitude) and more dispersive (change in wave frequency) than high-order ones and with low order schemes, attaining the same accuracy as with high order schemes needs many more grid points. Moreover, industrial configurations are complex, with many geometrical details. For such complex geometry, it is mandatory to consider unstructured mesh paradigms since mesh are generated more efficiently with unstructured approaches, even if the geometry is complex.

The industrial context is therefore the following: performing LES on complex geometry with unstructured grids and high-order schemes. Classical high-order approaches for unstructured grids introduce a polynomial reconstruction of data at any mesh interface. A high-order reconstruction needs a large stencil (more than 100 cells for a 5^{th} order WENO approach!) and this choice has a strong impact on High Performance Computing (HPC). It is clear that such an approach leads to many potential data exchanges in a parallel framework. A more appropriate solution consists in switching the considered paradigm.

New techniques based on high-order polynomial representation of data emerged during the last decades. The first one was the Discontinuous Galerkin (DG) approach from Reed and Hill [1], extended to the Navier-Stokes equations during the 2000's. The DG approach considers the Finite Element formalism inside any control volume and accounts for discontinuous data at mesh interface with the use of a Riemann solver. Recently, two other approaches were proposed. The first one, called Spectral Volume -SV- [2], consists in a polynomial

* Corresponding author. E-mail address: adrien.cassagne@cines.fr

representation of data from mean quantities computed over sub-cells: any control volume is decomposed in sub-cells and the mean quantity on each sub-cell is considered to define the high-order polynomial representation. In many aspects, the SV technique is an extension of the Finite Volume approximation. Both previous approaches need complex quadrature rules for integral computation and they are associated with a large cost per Degree of Freedom (DoF). In order to decrease the cost per DoF, the last technique, called the Spectral Difference approach -SD- [3,4] has been introduced. Equations are solved in their strong form, as in Finite Difference method and this point justifies the name of the approach. Later on, a comparison of several high-order approaches showed the superior efficiency of SD [5]. For these reasons, CERFACS decided to focus attention of the Spectral Difference method.

The principle of the SD method is to define a polynomial reconstruction of data inside each mesh cell and to account for the discontinuity of the reconstruction at element boundaries: the approach is locally continuous (inside the cell) and globally discontinuous. Several degrees of freedom are located in any control volume to define the polynomial and exchanges only concern the current cell and its direct neighbours, which leads to a limited stencil. The SD method performs on unstructured grid, which is a good way to overcome meshing constraints on complex geometry.

The development of our new solver JAGUAR [6] based on SD started two years ago with attention to HPC and on efficient algorithms. The goal was to work into two directions: serial efficiency and parallel efficiency. Obtaining a good efficiency for parallel computations is quite easy due to many local arithmetic operations and few exchanges. Moreover, high-order methods are generally assumed to be less efficient than low-order ones but, in our implementation, the same efficiency can be achieved. Finally, the last point of importance concerns the serial efficiency: Gflop/s measures obtained with JAGUAR are twice as high as those, obtained with standard CFD solvers (like elsA [7] or AVBP [8]). This high efficiency was achieved due to two reasons: first, the SD method needs a compact stencil, which is efficient for CPU cache, and moreover, a dedicated optimized task model has been implemented. JAGUAR has a quite simple structure and it is therefore a good candidate for tests on new architectures.

Achieved work and results

OpenMP code implementation and improvements

The first parallel version of the code considered the distributed memory paradigm with MPI. Then, multi-threading with OpenMP (shared memory paradigm) was implemented quickly, without analysing accurately the implementation practice on performance. The principle of the implementation was the following. Equations are advanced in time with the use of an explicit time integration algorithm: the solution is updated at each time step. The parallel performance is therefore linked with the treatment of the “right hand side” of equations, including the treatment of spatial terms.

In the first version, for each subroutine inside the time loop, there was a parallel zone creation (thread pooling zone): it is a way to get an OMP 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 “OMP 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 fact, some parts of the code are perfectly regular and it is easy to use a standard OMP 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 DoF (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 those 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. Fortran 90 encourages usage of short declarations for array initializations: we had to look for this kind of initializations in the solver and replace them by standard loops with OMP indices distribution. We also deleted some barriers between the subroutines in the solver by using the "no wait" clause where ever possible.

Threads binding and processes pinning strategies

This work was analysed on the Curie super-computer [9] 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 disable). There are NUMA (Non Uniform Memory Access) nodes: threads binding and processes pinning are very important because each socket possesses its own memory bank. It is more expensive to use a neighbour's memory bank than the direct memory bank.

In order to make a realistic comparison between the on-socket full OpenMP multi-threaded versions versus the on-socket full MPI version (for less or equal than 8 threads or less or equal than 8 processes), specific environment variables are set:

- `KMP_AFFINITY=granularity=fine,compact,1,0`: `KMP_AFFINITY` variable is only available with Intel compilers, it tells the OMP runtime to bind the running threads on the same socket,
- `I_MPI_PIN_PROCS=0,1,2,3,4,5,6,7`: `I_MPI_PIN_PROCS` variable specifies the core ids to use for MPI processes (here the core ids are all on the same socket).

When the number of threads or processes was higher than eight we have placed the 8 first threads or processes on the first socket and the remaining threads or processes have been placed on the second socket.

For hybrid computations (version of the code running with both MPI and OMP), we have observed that the optimal performance was achieved when we pinned one MPI process per socket (a total of 2 MPI processes with the chosen supercomputer with 2 sockets). In the configuration, each MPI process spawns 8 OMP threads. This is efficient because NUMA communication issues are avoided. Here, the environment variables configuration is:

- `I_MPI_PIN_DOMAIN=socket`: pins MPI processes to socket (process 0 is on socket 0, process 1 is on socket 1, process 2 is on socket 0, …),
- `OMP_NUM_THREADS=8`: spawn 8 OMP threads,
- `KMP_AFFINITY=granularity=fine,compact,1,0`: bind threads to one socket.

This configuration ensures that MPI processes are on separate sockets and that OMP threads are located in the same socket. This is efficient because there is only MPI communications between sockets and OMP threads can fully take advantage of the shared memory architecture.

Results

We used the Intel Fortran compiler (version 2013.2.146) because it shows better performance than GNU and PGI compilers (for our JAGUAR solver). For all the following experimentations, we used a 2D 432x432 mesh with a fourth degree polynomial interpolation (p=4) for each cell, leading to a fifth-order accurate solution.

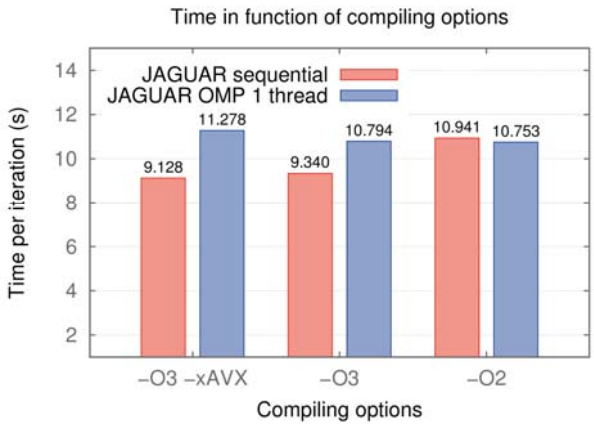

Figure 1: Restitution time depending on compiling options

First, we observed a very curious phenomenon when we compared the sequential version of the code (no MPI, no OMP) with the OMP version running with one thread (no MPI, `-openmp` flag at the compiling time). The same performance was expected but the experimentation demonstrated that the OMP version with one thread was significantly slower than the full sequential version (20% slower with `-O3 -xAVX` compiling options and 15% slower with `-O3` only). With the "`-O2`" compiling option, the problem disappeared (see Fig. 1).

We never succeeded in resolving this issue. We tried to profile the code with Intel VTune and we noticed that the compiled code was different: the sequential version performed more function inlining than the OMP version. We tried to force inlining in the OMP version, taking the fully sequential version as an example, but the OMP code was slower after the manual inlining. The problem is specific to two subroutines using the OMP tasking model so we assume this is linked to the tasks usage overhead.

Fair comparisons between full MPI, full OMP and hybrid MPI/OMP versions of the code are presented in the following results by considering all versions compiled with "`-O2`" option. We recall that if we had taken "`-O3`" as a standard then the full MPI version of the code would have been faster than other versions because of the higher performance on a single process (as presented above). In Fig. 2, we compared full OMP performance between the new version of JAGUAR (after all optimizations) and the old version.

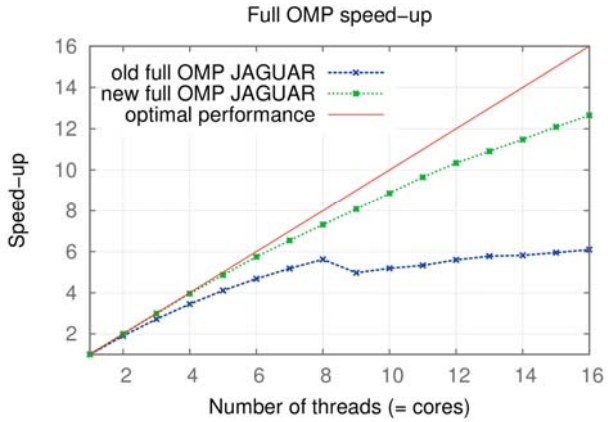

Figure 2: Full OMP comparison between old and new JAGUAR code

There was a large improvement of performances with the new version. In fact, the initial strategy (making parallel zones for each subroutine with a lot of thread synchronisations) is not scalable at all. Beyond 6 cores, the old JAGUAR solver is no more competitive with the new JAGUAR solver. We also noticed that the old JAGUAR solver paid the price of the NUMA architecture when there were more than 8 cores whereas the new JAGUAR solver seems to scale well until 16 cores. Considering this NUMA issue, we tried to launch a hybrid configuration of the old JAGUAR code (see Fig. 3).

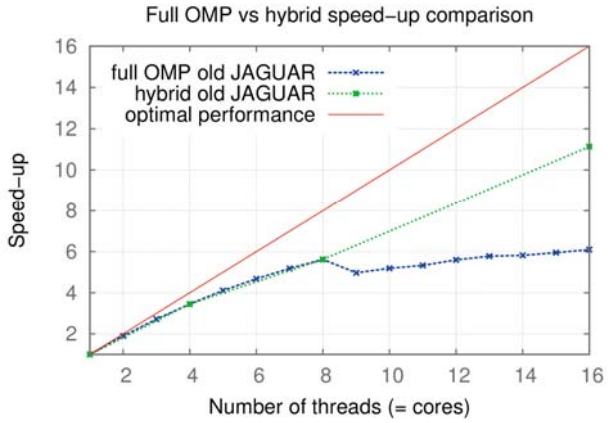

Figure 3: Full OMP versus hybrid on the old JAGUAR code

As we expected, the speed-up of the hybrid version of the old code is much better than the full OMP version and it confirms that the full OMP old version is not suitable for production. The same analysis with the new JAGUAR code enables to change drastically the conclusions (see Fig. 4).

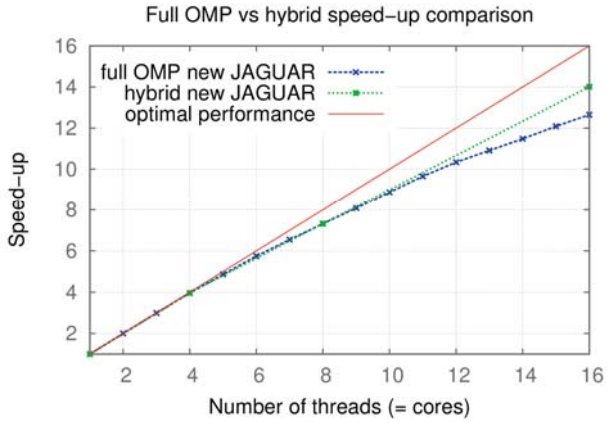

Figure 4: Full OMP versus hybrid on the new JAGUAR code

We achieved a 14-fold speed-up with the hybrid configuration and nearly a 13-fold speed-up with the full OMP new JAGUAR code. The hybrid version of the code is just a little bit better than the full OMP version but it is promising considering that we are using a dynamic OMP tasking model in a NUMA environment. We can now consider using the new full OMP JAGUAR code on a very large number of cores.
Fig. 5 shows a comparison between the previous hybrid version and the full MPI version of the code. The full MPI version is a little bit more efficient than the hybrid version (a 15-fold speed-up against a 14-fold speed-up). We did not succeed to beat the full MPI version with the hybrid one but we can expect that when there are more MPI processes, the number of MPI communications will grow and the hybrid version of the code will be able to handle this better.

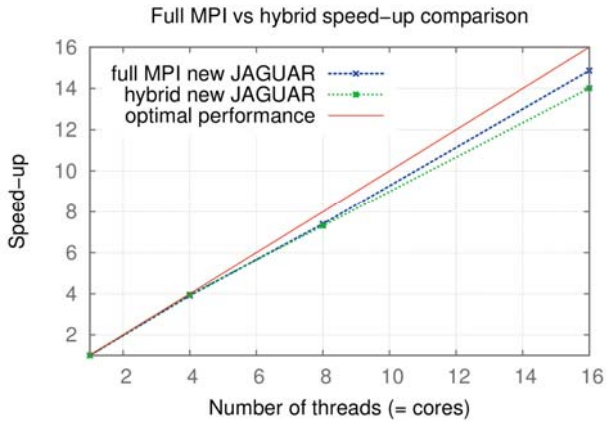

Figure 5: Full MPI versus hybrid version of code comparison

Fig. 6 demonstrated the gap between the initial version of the code and the optimized one (for 64 cores, 8 sockets, 4 nodes). We compared the hybrid configurations because for both, old JAGUAR code and new JAGUAR code, the more efficient configurations were obtained using OMP. We achieved a 54-fold speed-up in the new version against a 44-fold speed-up in the old version. These are promising results but we were unable to launch JAGUAR on more cores (we had some problems to generate the big meshes required for many core computations and we succeeded too late to achieve suitable OMP and hybrid performance).

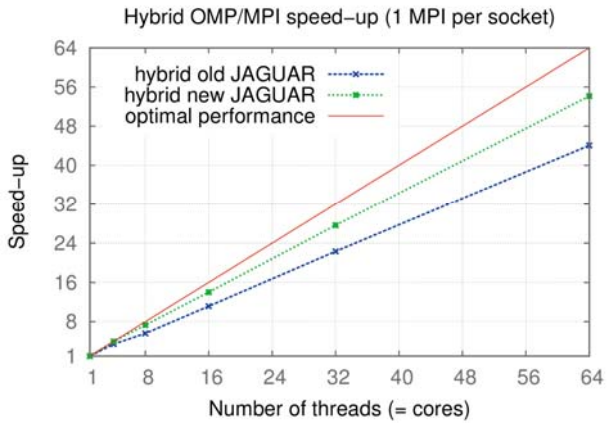

Figure 6: Hybrid OMP/MPI speed-up for 4 nodes (or 8 sockets)

Instead of launching hybrid JAGUAR code on a big number of cores, we launched (at the beginning of the project) a full MPI version on 512 cores (Fig. 7). The results of this full MPI version were encouraging and we could estimate the same performances for the hybrid version of the code. However, we also noticed that the performance was not as good as we expected for 512 cores: this was due to an increase in communications and a decrease in computations per core. If we had used a bigger mesh, the speed-up for 512 cores would have been better.

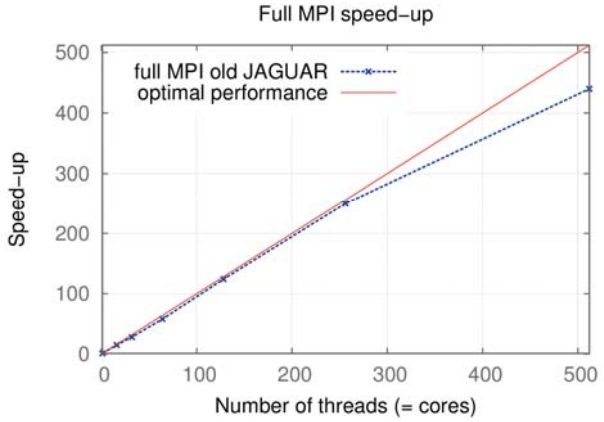

Figure 7: Full MPI old JAGUAR code on 512 cores

Lessons learned and future work

The goal of the project was the modification of a CFD solver with a new discretization paradigm in order to track the best possible performance with hybrid OMP/MPI paradigm. The initial version of JAGUAR was efficient for parallel MPI communications but the OMP paradigm was not implemented correctly in order to attain a good performance.

During the project, we succeeded in creating a hybrid version of the JAGUAR code and attention was paid on the OMP implementation in order to achieve good performance. The easiest implementation (put some directives without modifying too much the code) was not enough and we had to think carefully about the cost of thread synchronisations in order to reach satisfactory results.

Even if the code is well written, threads binding and processes pinning are very important. This can be made at run time by using environment variables. It is crucial to understand how to bind threads on NUMA architectures because the cost of the communications between the different sockets is much higher than on the socket itself.

In the near future, it is planned to generate bigger meshes in order to launch the hybrid JAGUAR configuration on more cores. The presented results show a really good strong scalability until 512 cores on a small mesh. On a large number of cores, we expect to outperform the full MPI version with the hybrid version.

Moreover, we think that the current OMP code is a good initial version for performance analysis on the Many Integrated Core (MIC) architecture like Xeon Phi. Of course, we are aware that there will be some improvements to make if we want to reach high performance.

References

[1] W.H. Reed and T.R. Hill, Triangular mesh methods for the neutron transport equation, *Los Alamos National Laboratory, New Mexico, USA, Tech. Report LU-UR-73-279*, **1973**.

[2] Z.J. Wang, Spectral (finite) volume method for conservation laws on unstructured grids: Basic formulation, *Journal of Computational Physics*, **2002**, 178, 210-251.

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

[4] Y. Sun, Z. J. Wang, and Y. Liu. High-order multidomain spectral difference method for the Navier-Stokes equations on unstructured hexahedral grids, *Communication In Computational Physics*, 2(2):310-333, **2007**.

[5] Yu, M. & Wang, Z.J., On the accuracy and efficiency of several discontinuous high-order formulations, *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, 7-10 January, Grapevine (Dallas/Ft. Worth Region), Texas, AIAA Paper 2013-0855, **2013**.

[6] JAGUAR code web page: *http://www.cerfacs.fr/~puigt/jaguar.html*.

[7] elsA web page: *http://elsa.onera.fr*.

[8] AVBP web page: *http://www.cerfacs.fr/4-26334-The-AVBP-code.php*.

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

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.