11th World Congress on Computational Mechanics (WCCM XI)
5th European Conference on Computational Mechanics (ECCM V)
6th European Conference on Computational Fluid Dynamics (ECFD VI)
E. Oñate, J. Oliver and A. Huerta (Eds)

# DIRECT NUMERICAL SIMULATION OF TURBULENT FLOWS WITH PARALLEL ALGORITHMS FOR VARIOUS COMPUTING ARCHITECTURES

**A. GOROBETS[1,2], F.X. TRIAS[1], R. BORRELL[3], G. OYARZÚN[1] AND A. OLIVA[1]**

[1] Heat and Mass Transfer Technological Center, ETSEIAT, Technical University of Catalonia
C/ Colom 11, 08222, Terrassa, Spain, e-mail: cttc@cttc.upc.edu, http://cttc.upc.es/

[2] Keldysh Institute of Applied Mathematics of Russian academy of sciences
4A, Miusskaya Sq., Moscow, 125047, Russia

[3] Termo Fluids, S.L., c/ Magí Colet 8, 08204 Sabadell (Barcelona), Spain, http://termofluids.com/

**Key Words:** *Parallel CFD, turbulent flows, MPI, OpenMP, OpenCL.*

**Abstract.** The purpose of the work is twofold. Firstly, it is devoted to the development of efficient parallel algorithms for large-scale simulations of turbulent flows on different supercomputer architectures. It reports experience with massively-parallel accelerators including graphics processing units of AMD and NVIDIA and Intel Xeon Phi coprocessors. Secondly, it introduces new series of direct numerical simulations of incompressible turbulent flows with heat transfer performed with the considered algorithms.

## 1 INTRODUCTION

The new trends in the evolution of high performance computing (HPC) systems require more and more scalable algorithms with higher degree of parallelism and compatibility with fundamentally different parallel models. The modern supercomputers can be classified into three kinds: 1) classical ones that run on computing power of central processing units (CPU); 2) hybrid machines with CPUs and graphics processing units (GPU); 3) hybrid machines with CPUs and Intel Xeon Phi accelerators of many integrated core (MIC) architecture.

The first type, the basic one, requires highly scalable parallel algorithms that can run on thousands of cores. It also needs efficient shared-memory parallelization with large number of threads to engage modern multi-core nodes: two 12-core Intel Xeon CPUs with Hyper Threading (HT) can execute 48 parallel threads on a dual-CPU node. In addition it needs efficient vectorization since AVX extension operates with vectors of 4 doubles. The second type requires adaptation of algorithms to the streaming processing which is a simplified form of parallel processing related with SIMD (single instruction multiple data) model. This can be a challenge itself. The third type requires much more deep multi-threaded parallelism and vectorization than the first type. Furthermore, there are ARM-based architectures [1] that combine complexity of abovementioned types with weakness of simplified RISC processors. And of course all of those hybrid types require a lot of attention to the memory access and complex load balancing between CPU and accelerators.

The only framework that allows using all of these kinds of computing equipment appears to be the open standard OpenCL (Open computing language) [2]. The parallel programming paradigm that fits all of these kinds is the simplest one, the streaming processing. It assumes processing of a large number of similar independent tasks.

Parallel finite-volume CFD algorithms for modelling of turbulent flows are considered in this work. A multilevel approach that combines different parallel models is used. MPI is used on the first level within the distributed memory model to couple computing nodes of a supercomputer. On the second level OpenMP is used to engage multi-core CPUs and/or Intel Xeon Phi accelerators. The third level exploits the computing potential of massively-parallel accelerators. The hardware independent OpenCL standard is used to compute on accelerators of different architectures within a general model for a combination of a central processor and a math co-processor. The algorithms under consideration in their "classical" MPI+OpenMP parallel implementation, including [3,4], were designed to scale till the range of up to 100000 CPU cores. The presented here OpenMP and OpenCL-based extensions have been developed in order to exploit the computing potential of modern hybrid machines. This work summarizes our experience in adapting the computational algorithms to different accelerator architectures. The specific scheduler infrastructure [5] for automated management of OpenCL tasks has been chosen to simplify the implementation of MPI and host-accelerator communications overlapped with computations.

Finally, a summary of our recent direct numerical simulations (DNS) performed using the high-order finite-volume symmetry-preserving spatial discretization [6] is presented. The cases include a natural convection flow inside a closed air-filled (Prandtl number 0.71) differentially heated cavity of height aspect ratio 3.84 and depth aspect ratio 0.86. The Rayleigh number (based on the cavity height) is $1.2\times10^{11}$. This configuration resembles the experimental set-up performed by D. Saury et al. [7]. Another case is a flow around a square cylinder at Reynolds number 22000 (based on the height of the obstacle). The flow has been computed using the 4-th order scheme on a grid with around 300 million of nodes.

## 2 SIGNIFICANTLY MULTI-THREADED PARALLELIZATION

### 2.1 Parallel algorithm for structured meshes

First let us consider the algorithm on structured meshes for incompressible flows. Its basic OpenMP parallelization [3] was intended for running with 10-20 threads per MPI process. The algorithm is based on explicit time integration scheme with the classical fractional step projection method [8] used to handle pressure-velocity coupling: a predictor velocity is explicitly evaluated without considering the contribution of the pressure gradient. Then by imposing the incompressibility constraint it leads to a Poisson equation for pressure to be solved on each time-step. Basically the algorithm is composed of two parts: "explicit" and "implicit". Implicit part is the Poisson equation solution stage, while explicit part includes all the rest: momentum and energy transfer equations, calculation of pressure gradient, etc. Implementation of the algorithm is composed of six basic operations: 1) linear operator (diffusion); 2) nonlinear operator (convection); 3) vector operations (summations, multiplication by constant, etc); 4) FFT (and inverse FFT) transform; 5) sparse matrix-vector product; 6) dot products, norms. The last three operations compose the implicit part. Operations of types 1), 2), 3), 4) are represented as two or three nested loops over two or three

spatial directions, respectively. The original OpenMP parallelization [3] consisted in decomposing the outer loop between parallel threads in those operations. This approach is of course limited by the number of mesh nodes in one spatial direction in each MPI subdomain. This number (typically varies from 40 to 100) is generally not enough to occupy 240 parallel threads on the Intel Xeon Phi accelerator. The solution is straightforward: workload is decomposed in the two spatial directions. The limits of two outer loops are recalculated in an optimal way giving for each thread an approximately equal nonintersecting workload.

The Poisson solver for problems with one periodic direction is based on uncoupling the original 3D system by means of FFT diagonalization into a set of independent 2D systems, planes in other words, which are as many as the number of nodes in the periodic spatial direction. The original OpenMP parallelization was just sharing planes between threads. Each thread was solving its set of planes by means of a preconditioned conjugate gradient solver [9] so each plane was processed by no more than one thread. This approach was also insufficient for hundreds of threads since typically there isn't that much planes. Parallelization of the solution of a 2D system was required. Block LU preconditioner (LU factorization is built for local matrices of MPI subdomains or even smaller blocks discarding interface between the blocks) did not fit well for such a parallelization and it has been replaced with the sparse approximate inverse (SAI) [10].

Its parallelization is straightforward since it is composed of two sparse matrix vector products, by the two approximations of the Cholesky factor inverses. The optimal SAI configuration chosen is level 3 (using the sparsity pattern of $A^3$, where A is the system matrix, to evaluate the inverse approximates) filtered to the tolerance of 0.15 with respect to the diagonal entries in each row. In tests with 64 subdomains for a 100x200 plane the resulting computing price of solution with SAI is only 5% higher and the number of iterations is 22% more compared to the block LU one, while SAI is much more efficient on accelerators.

The abovementioned modifications adapt the algorithm for a significantly multithreaded OpenMP parallelization resulting in rather high internal speedups inside Intel Xeon Phi (up to 200 times compared to a sequential execution on the same accelerator).

### 2.2 Unstructured meshes

Let us consider discretizations [4,11] on unstructured meshes. A high-order finite-volume approach consists in calculating fluxes through control volume faces, either with edge-based reconstruction [12] or polynomial reconstruction [11], summating fluxes into nodes (or cell centers). Dissipative terms can be discretized using finite-element approach [13].

The algorithm consists of basic operations like calculation of fluxes through faces, dissipative term contribution (viscosity) which is mainly calculation of gradients on mesh elements, boundary conditions, source terms, etc. The main problem for parallelization is to avoid data interdependency in operations over computing domain elements of one kind with modifying data in elements of another kind. For example, calculation of fluxes consists in an operation over a set of faces with writing result into set of cells. The operation is composed of: reconstruction of values in centers of faces from values in centers of cells [11,12]; taking values from "left" and "right" sides restoring resulting flux through the faces [14,15]; accumulating total fluxes in cells. The loop over faces that represents this operation cannot be directly decomposed between threads since processing faces of the same cell simultaneously

will result in error. The same situation appears in another kind of operation, calculation of nodal gradients in a vertex-centered approach: gradients on tetrahedrons are calculated in a loop over tetrahedrons and summated to the nodes (vertices) of tetrahedrons. Processing simultaneously tetrahedrons that share the same node can result in error.

Generally speaking there are elements of computing domain (e.g. mesh elements – tetrahedrons, hexahedrons, edges or faces of cells, etc.) that connect cells in cell-centered approach or nodes/vertices in vertex-centered approach and there is data interdependency between elemental operations (e.g. on modifying data in one cell by more than one thread). Different straightforward strategies can be used to eliminate this data interdependency.

1) Decomposition of an operation into stages using auxiliary arrays over mesh elements to store the intermediate values. This way the values are first computed in parallel in a loop over mesh elements and then these values are summated into cells in parallel in a loop over cells using mesh connectivity graph.
2) Dividing the set of mesh elements into subsets in such a way than in each subset there are no elements sharing same cell. This way each subset can be processed in parallel without fear of intersection and synchronization points are needed between subsets.
3) Second level decomposition. Each MPI subdomain is decomposed further into parts of OpenMP threads. Then each thread process only those mesh elements that have at least one node/cell belonging to the thread. The threads write results only to their nodes/cells. This way there is no intersections but there is some extra computations for interface elements that have nodes belonging to different threads.

Approach 1) fits the streaming processing, simplified parallel paradigm, and it is valid for any kind of accelerator. For example this way calculation of fluxes is divided into two stages: firstly, calculate fluxes through faces in a loop over faces and store result in an intermediate array over faces; secondly, summate fluxes from faces to cells in a loop over cells using inverse mesh topology that stores for each cell the list of its faces. This results in perfectly parallelizable and completely independent operations.

However, this approach cannot be used when the amount of intermediate data is too big. For example, in the calculation of nodal gradients for high-order edge-based scheme [12] it would need to store up to 45 values for each tetrahedron (15 gradients in 3 spatial directions). That the number of tetrahedrons is around 6 times the number of nodes. It is like 270 arrays over nodes, which is inacceptable. In this case option 2) can be used. It fits streaming processing parallel model and can run on any kind of accelerator. However, its drawback is some inefficiency in memory access pattern: such a decomposition of mesh elements into subsets prevents from coalescing of memory access and re-usage of values from neighbor cells since no elements sharing same cell are allowed in a subset.

The option 3) with 2-level decomposition ($1^{st}$ level – MPI, $2^{nd}$ level – OpenMP) works well only for a small number of threads like 8-16. Interface elements that connect nodes (or cells) from different threads are processed more than once: each element is processed by all the threads whose nodes it affects. With a lot of threads this overhead on processing interface elements becomes inacceptable.

This option can be extended with another level of decomposition in order to adapt to hundreds of threads. An MPI subdomain is decomposed further using the same mesh partitioning tool into as many parts as the number of threads. It is a 2-nd level decomposition. Mesh elements belong to 2-nd level interface if they connect cells from different parts, other

elements are 2-nd level inner. The set of interface elements with nodes/cells it includes can be decomposed further in the same way. A connectivity graph is built for the 2-nd level interface elements only and it is decomposed with the same partitioning tool. Doing so, we get the 3-rd level interface set, as it shown on Figure 1. Now the operation over mesh elements can be performed in three stages with synchronization points in between: parallel processing of 2-nd level inner elements; parallel processing of 3-rd level inner elements; parallel processing of 3-rd level interface elements with writing results only to cells that belong to the thread. So we have moved some computing overhead to a much smaller set of elements allowing several hundreds of threads. For example, decomposing a mesh of $1.2\times10^5$ nodes, $3.7\times10^5$ faces, $6.3\times10^5$ tetrahedrons for 240 threads gives 2nd level interface of $5.2\times10^4$ faces and $1.6\times10^5$ tetrahedrons. The 3rd level interface reduces to only $4.5\times10^3$ faces $2.6\times10^4$ tetrahedrons.

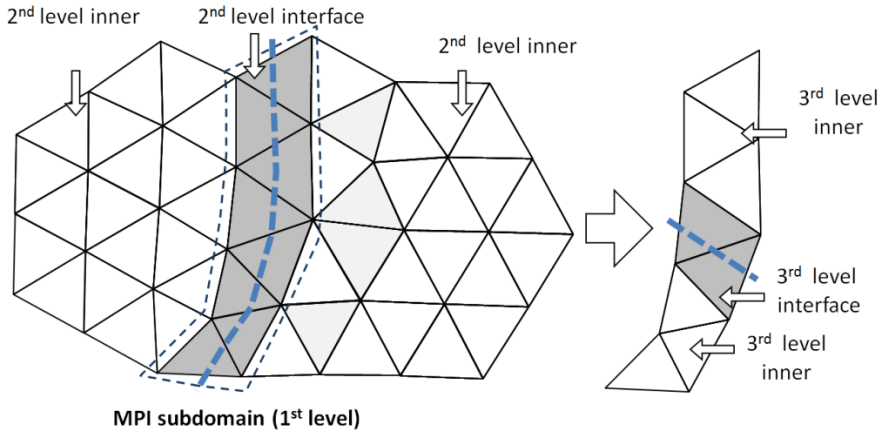


**Figure 1**: Distributing triangles (2D vertex-centered scheme) in two levels of OpenMP inside MPI subdomain

Similar approach, even more generalized, is used in [16] where mesh elements are grouped into subsets using some graph coloring algorithm and these subsets are also grouped into sets of subsets in a way that in each set of subsets there is no data intersection between subsets. In each set of subsets, the subsets can be processed by different threads simultaneously. Synchronization points are needed between sets of subsets. Finally, running 240 threads on 60 cores with 4-thread HT of Intel Xeon Phi also showed speedups around 150–200 times for different operations compared to the sequential execution on the same accelerator.

## 3 OVERLAP OF COMMUNICATIONS AND COMPUTATIONS ON GPU

Overlap of computations and communications means execution of MPI and/or host-device data transfer simultaneously with computations on accelerators. To do so there must be independent communication and computation tasks. The computing domain is decomposed into subdomains of MPI processes or subdomains of accelerators in order to distribute the workload between multiple accelerators or supercomputer nodes. Communications are needed in particular in the halo update operation that is required on each time step to transfer the values in the interface elements between accelerators. Data transfer includes communication between the host and accelerators inside a computing node and MPI communications between different processes. Implementation of such an approach is described in detail on example of a sparse matrix-vector product in [17].

Let us consider an operation over a set of elements of some type. The set of elements of a subdomain is decomposed into the inner and interface parts. The inner set contains only the elements that are connected (e.g. coupled by a stencil of spatial discretization scheme) with cells belonging to this subdomain. Interface elements are those connected with cells from other subdomains. The operation is divided into two stages: processing of inner elements and processing of interface elements. Only the second stage needs data exchange with neighboring subdomains, hence, communications can be executed simultaneously with the first stage.

To simplify implementation of overlapped communications an automated scheduler infrastructure for OpenCL tasks [5] has been used. In brief, the concept of the scheduler is based on two definitions: register and instruction. Register is an arbitrary linear region in a device global memory. Instruction is an arbitrary compute kernel on a device. The device can be interpreted as a virtual machine which runs instructions on registers. The devices are operated by a special scheduler that manages the available resources, and provides a developer with abilities to control the registers, to transfer data between host and devices, and to launch compute kernels on devices. Instructions and dependencies between them form a dependency graph, which is a program for the scheduler. There are three types of instructions considered: data transfer, kernel execution and auxiliary commands. In more details the scheduler and the overall software infrastructure are described in [5]. The Scheduler relies on the instruction dependency graph and provides the data transfer overlap automatically.

Performance of overlapped communications has been demonstrated on example of a first-order algorithm with a minimal computing price. The test has been performed on the K100 supercomputer that consists of 64 nodes. Each node has two 6-core Intel Xeon X5670 2.9GHz CPUs and three NVIDIA C2050 GPUs. An example of the strong speedup is shown on Figure 2. It can be seen that reasonable efficiency is kept until the load of around 0.2 million of cells per GPU, then efficiency drops down, however, the code still keeps accelerating.

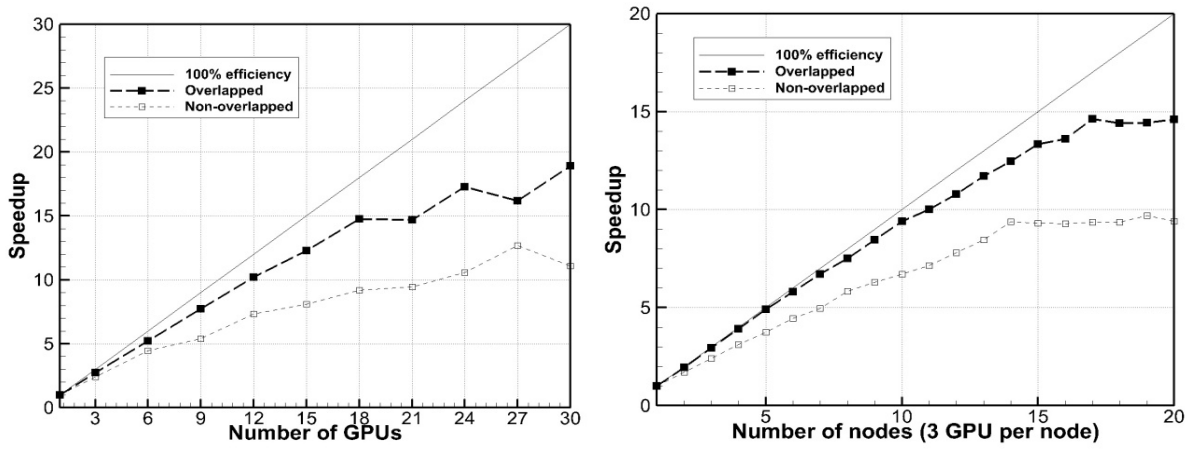


**Figure 2**: Speedups on K100 supercomputer. Left – starting from one GPU for a mesh with 4 millions of cells. Right – starting from one node with 3 GPUs for a mesh with 16 millions of cells

## 4 PERFORMANCE SUMMARY

For the OpenCL implementation of the presented algorithm on structured meshes and its performance on GPU the reader is referred to [18]. The acceleration on a GPU compared to a

multi-core CPU overcame the power consumption ratio of the devices for all the GPUs considered (both AMD and NVIDIA). Details on OpenCL implementation and performance for the algorithm on unstructured meshes can be found in [11]. Comparison of performance of this algorithm is shown for a variety of computing devices on Figures 3 and 4. It can be noted that for the 1-st order scheme (Figure 3) NVIDIA GTX TITAN outperforms AMD 7970 while for the 2-nd order polynomial-based scheme which requires much more resources (registers and shared memory usage) AMD one significantly outperforms NVIDIA one. This indicates the insufficiency of register and local memory of NVIDIA architecture that prevents from achieving high occupancy of the device and reduces efficiency.

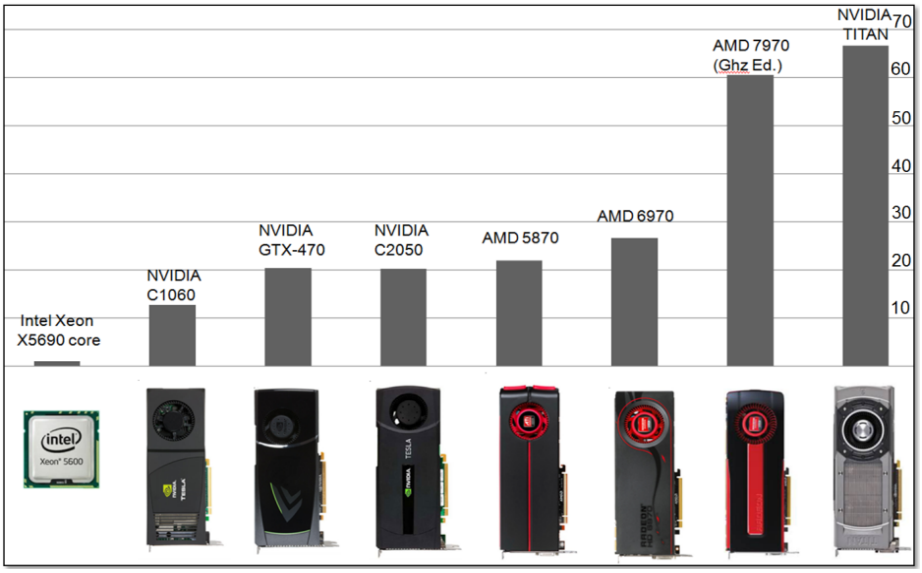


**Figure 3**: Comparison of performance on a mesh with 472114 cells (flow around a sphere) for different devices using a 1-st order finite-volume scheme for unstructured meshes

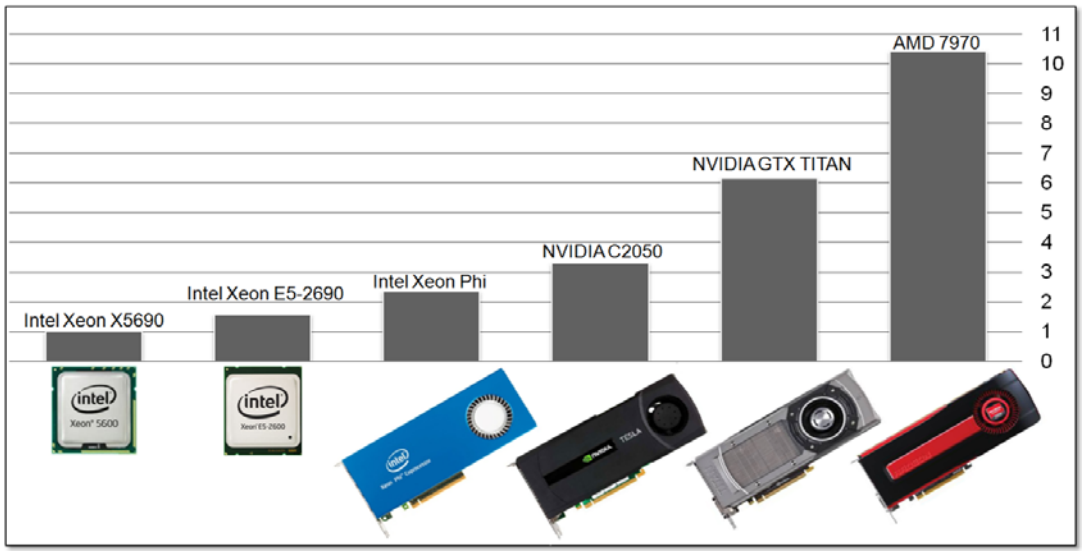


**Figure 4**: Comparison of performance on a mesh with 679339 cells (flow around a sphere) for different devices using a 2-nd order polynomial-based finite-volume scheme for unstructured meshes

Another conclusion that comes from Figure 4 is that Intel Xeon Phi architecture appears to be weaker than GPUs of both AMD and NVIDIA. As it was an OpenCL implementation one may blame the Intel OpenCL drivers for the Intel Xeon Phi which are not yet mature enough. However, OpenMP implementations show similar behavior resulting in no more than 10%-20% speedup compared to 8-core Intel Xeon E5-2690 CPU. So the common statement that Intel Xeon Phi is much easier to use than GPU because it can handle the same CPU code is an illusion. The computing power of this kind of accelerator is much more difficult to get.

Regarding OpenMP implementations there is an interesting behavior on Intel Xeon Phi accelerator for both algorithms on structured and unstructured meshes: It demonstrates rather high internal speedups (up to 200) compared to a sequential execution on the same accelerator. However, speedup compared to 8-core CPU appeared to be insignificant (10-20%). This result is surprising and needs further study. Poor vectorization could be a reason but it would also affect performance on CPU as well since its performance also relies a lot on SIMD extensions. Another reason which seems more realistic can be related to insufficient memory latency hiding mechanisms that are based on 4-thread hyper threading (GPU can have tens of threads switching for latency hiding). Perhaps suspicions about poor cache performance of the accelerator can also explain this situation.

## 5 OVERVIEW OF DIRECT NUMERICAL SIMULATIONS

This section intends to illustrate the application area of the algorithms and to inform the reader on the new DNS cases to be published soon in more detail. The simulations performed using the high-order finite-volume symmetry-preserving spatial discretization [6].

The first case is an air-filled (Pr = 0.7) differentially heated cavity of height aspect ratio 3.84 and depth aspect ratio 0.86. The Rayleigh number (based on the cavity height) is $1.2\times10^{11}$. This configuration resembles the experimental set-up performed by D. Saury et al. [7]. The cavity is subjected to a temperature difference across the vertical isothermal walls whereas the “Fully Realistic” boundary conditions proposed in [19] are imposed at the rest of walls. They are time-independent analytical functions that fit their experimental data. The simulation has been carried out on a grid with 48 millions of nodes using MVS-10P supercomputer (JSCC RAS, Russia). The time-averaged temperature field displayed in Figure 5 (right) shows that the cavity is almost uniformly stratified. A dimensionless stratification of $S \approx 0.45$ obtained is in a rather good agreement with the experimental results obtained by D. Saury et al. [7] ($S = 0.44 \pm 0.03$ for wall emissivity 0.1). Detailed flow statistics including heat transfer analysis, turbulent statistics, comparison with experimental results and LES is in the near future plans for publication.

The second case is the flow around a square cylinder at Reynolds number 22000 (based on the diameter of the cylinder, D, and the inflow velocity). This configuration has been widely studied to test the performance of numerical methods and turbulence models. However, most of the numerical studies have been performed using RANS and LES modeling techniques whereas the attempts of DNS simulations are limited to relatively ’coarse’ meshes. Hence, the experimental results by Lyn et al. [21] have been usually taken as the reference solution. However, since this flow configuration is used for benchmarking purposes the availability of DNS results is of extreme importance. The flow has been computed using the fourth-order symmetry-preserving scheme [6] on a grid with around 300 millions of nodes on 800 CPU

cores of MareNostrum supercomputer (BSC, Spain). An illustrative snapshot showing the near wake vortex structures is displayed in Figure 6. Moreover, direct comparison with the experimental results by Lyn et al [21] and Minguez et al [22] is shown in Figure 7.

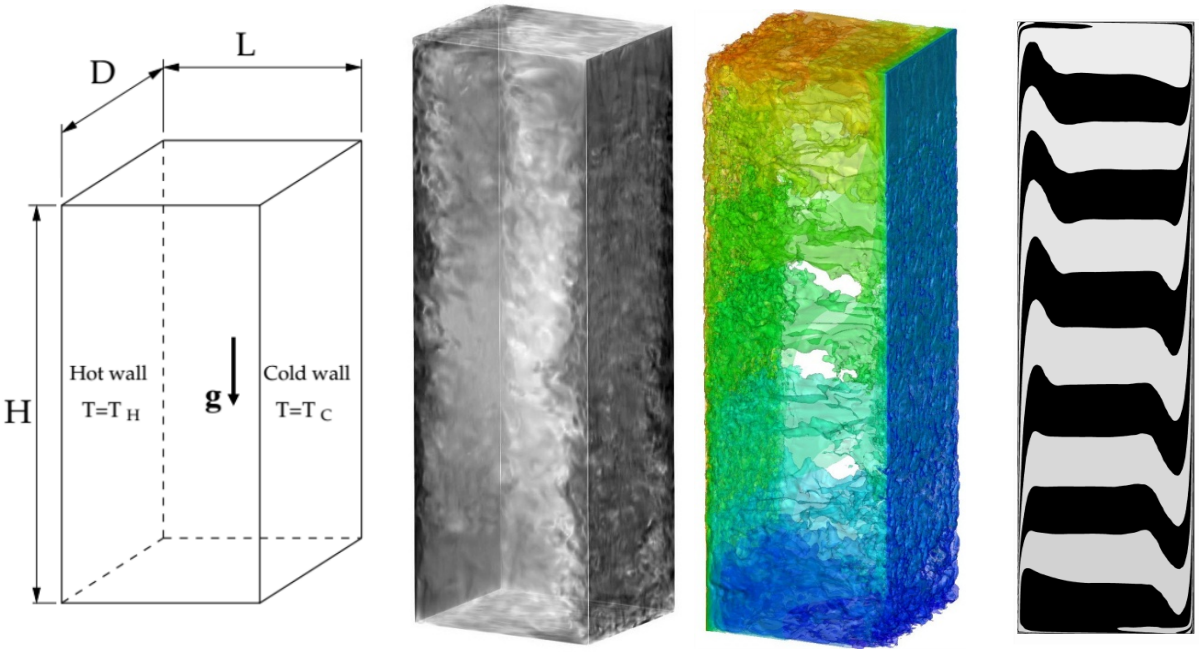


**Figure 5**: Differentially heated cavity. From left to right: schema of domain; instantaneous field of velocity module; instantaneous isotherms, averaged temperature field at the cavity mid-depth

## 6 CONCLUSIONS

Parallel finite-volume algorithms for large-scale simulations of turbulent flows have been considered. Simple approaches for OpenMP parallelization aimed at hundreds of threads have been presented for both structured and unstructured meshes. High internal speedups ensuring good level of parallelism demonstrated on Intel Xeon Phi surprisingly coexist with relatively poor net performance comparable with 8-core CPU. Implementation of overlapped communications with an OpenCL task scheduler infrastructure has demonstrated promising results. Performance comparison for basic OpenCL kernels of the algorithm on unstructured meshes showed that the different GPUs considered substantially outperform Intel Xeon Phi accelerator. Also, the AMD GPU tends to be more efficient than NVIDIA on heavy computing kernels.

## ACKNOWLEDGEMENTS

This work has been financially supported by the Ministerio de Ciencia e Innovación, Spain: ENE2010-17801 and a Ramón y Cajal postdoctoral contract RYC-2012-11996. Calculations have been performed on the IBM The supercomputers MareNostrum of BSC, Spain, MVS10P of JSCC RAS, Russia, and K100 of KIAM RAS, Russia, have been used for our calculations. The authors thankfully acknowledge these institutions.

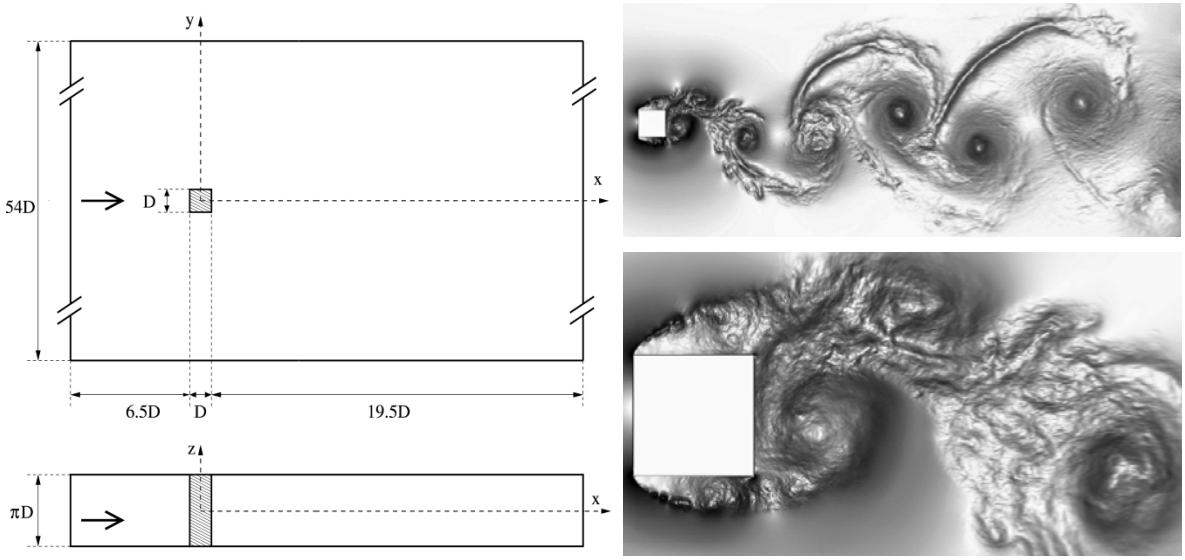


**Figure 6**: Flow around a square cylinder at Re = 22000. Schema (left) and near wake vortex structures (right)

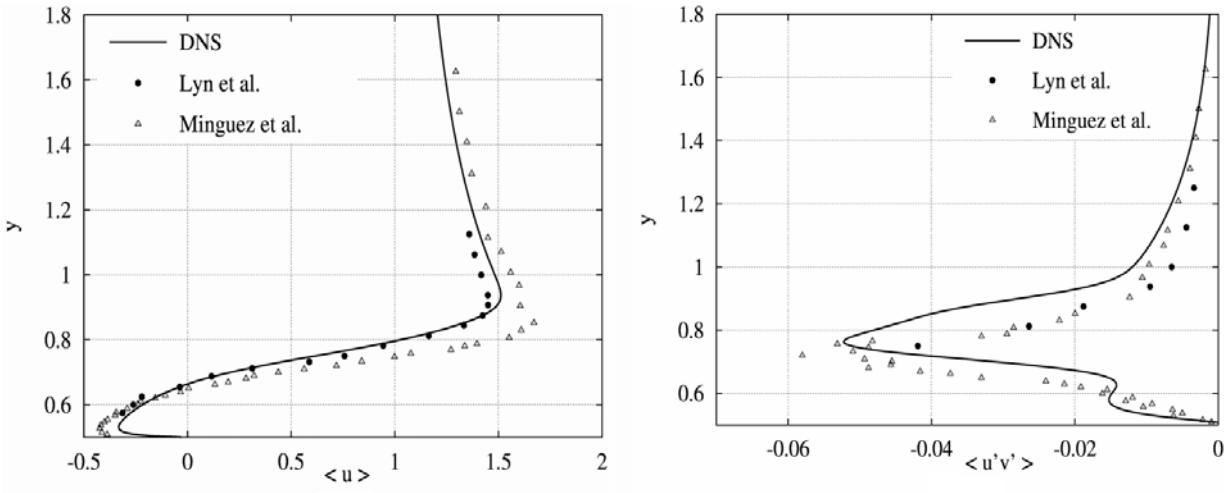


**Figure 7**: Comparison with the experimental results by Lyn et al [21] and Minguez et al [22] at x/D=0.125.

## REFERENCES

[1] Ramirez A. The Mont-Blanc architecture. *International Supercomputing Conference (ISC 2012)*, Hamburg, Germany, June 17–21, 2012.

[2] Khronos Group, The OpenCL Specification, Version: 1.1, 2010; Version 1.2, 2011; Version 2.0, 2013; http://www.khronos.org/opencl/

[3] A. Gorobets, F. X. Trias, R. Borrell, O. Lehmkuhl, A. Oliva, "Hybrid MPI+OpenMP parallelization of an FFT-based 3D Poisson solver with one periodic direction", *Elsevier, Computers & Fluids*, 49 (2011), pp. 101-109.

[4] Abalakin I.V., Bahvalov P.A., Gorobets A.V., Duben A.P., Kozubskaya T.K., Parallel code NOISETTE for large-scale CFD and aeroacoustics simulations, *Computing methods and programming*, vol.13 (2012), pp. 110-125.

[5] Bogdanov P. B., Efremov A. A., Programming infrastructure of heterogeneous computing based on OpenCL and its applications. *GPU Technology Conference GTC-2013*, March 18-21, San Jose, California, USA.
[6] R. W. C. P. Verstappen and A. E. P. Veldman. Symmetry-Preserving Discretization of Turbulent Flow. *Journal of Computational Physics*, 187:343-368, 2003.
[7] D. Saury, N. Rouger, F. Djanna, and F. Penot. Natural convection in an air-filled cavity: Experimental results at large Rayleigh numbers. *International Communications in Heat and Mass Transfer*, 38:679–687, 2011.
[8] Chorin AJ. Numerical solution of the Navier–Stokes equations. *Journal of Computational Physics,* 1968; 22:745–62.
[9] Saad Y. Iteratives methods for sparse linear systems. Philadelphia (PA): SIAM; 2003.
[10] Chow E, Saad Y. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM J Sci Comput* 1998;19:995–1023.
[11] S. A. Soukov, A. V. Gorobets, P. B. Bogdanov, Adaptation and optimization of basic operations of a CFD algorithm on unstructured meshes to compute on massively-parallel accelerators, *Comput. math and math physics*, 2012, vol. 53, #8, pp 1360–1373.
[12] Ilya Abalakin, Pavel Bakhvalov, Tatiana Kozubskaya. Edge-Based Methods in CAA. – *In "Accurate and Efficient Aeroacoustic Prediction Approaches for Airframe Noise", Lecture Series* 2013-03, Ed. by C.Schram, R.Denos, E.Lecomte, von Karman Institute for Fluid Dynamics, (2013) (ISBN-13 978-2-87516-048-5).
[13] C. A. J. Fletcher, Computational Galerkin Methods, *Springer Series in Computational Physics,* 1984.
[14] Roe P.L. Approximate Riemann Solvers, Parameter Vectors and Difference Schemes, *J. Comput. Phys.* 1981. 43, N2. 357-372.
[15] Huang L.C. Pseudo-unsteady difference schemes for discontinuous solution of steady state one-dimensional fluid dynamics problems, *J. Comp. Phys*. 1981. 42, N1, 195-211.
[16] Aubry R., Houzeaux G., Vazquez M., Cela J. M. Some useful strategies for unstructured edge-based solvers on shared memory machines, *International Journal for Numerical Methods in Engineering,* 2010, vol. 85, pp. 537–561.
[17] G. Oyarzun, R. Borrell, A. Gorobets, A. Oliva, MPI-CUDA sparse matrix–vector multiplication for the conjugate gradient method with an approximate inverse preconditioner, *Computers and Fluids*, 92 (2014), pp. 244–252.
[18] A. Gorobets, F.X. Trias, A. Oliva, A parallel MPI+OpenMP+OpenCL algorithm for hybrid supercomputations of incompressible flows, *Computers and Fluids*, 2013,Volume 88, 15 December 2013, Pages 764–772
[19] A. Sergent, P. Joubert, S. Xin, and P. Le Quéré. Resolving the stratification discrepancy of turbulent natural convection in differentially heated air-filled cavities. Part II: End walls effects. *International Journal of Heat and Fluid Flow*, 39:15–27, 2013
[20] D. Lyn, S. Einav, W. Rodi, and J. Park. A laser doppler velocimetry study of ensemble-averaged characteristics of the turbulent near wake of a square cylinder. *Journal of Fluid Mechanics*, 304:285–319, 1995.
[21] M. Minguez, C. Brun, R. Pasquetti, and E. Serre. Experimental and high-order LES analysis of the flow in near-wall region of a square cylinder. *International Journal of Heat and Fluid Flow*, 32(3):558–566, 2011.