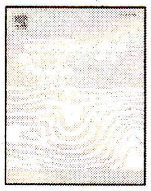
Contents lists available at ScienceDirect

Computers and Fluids

journal homepage: www.elsevier.com/locate/compfluid



HPC^2—A fully-portable, algebra-based framework for heterogeneous computing. Application to CFD

Q2 X. Álvarez[a,*], A. Gorobets[a,b], F.X. Trias[a], R. Borrell[a,c], G. Oyarzun[a]

[a] Heat and Mass Transfer Technological Center, Technical University of Catalonia, C/ Colom 11, Terrassa (Barcelona) 08222, Spain
[b] Keldysh Institute of Applied Mathematics RAS, Miusskaya Sq. 4, Moscow 125047, Russia
[c] Termo Fluids, S.L., C/ Magí Colet 8, Sabadell (Barcelona), 08024, Spain

ARTICLE INFO

Article history:
Received 2 November 2017
Accepted 23 January 2018
Available online xxx

Keywords:
Heterogeneous computing
MPI+OpenMP+OpenCL
Hybrid CPU+GPU systems
CFD
Symmetry-preserving discretization

ABSTRACT

The variety of computing architectures competing in the exascale race makes the portability of codes of major importance. In this work, the HPC^2 code is presented as a fully-portable, algebra-based framework suitable for heterogeneous computing. In its application to CFD, the algorithm of the time-integration phase relies on a reduced set of only three algebraic operations: the sparse matrix-vector product, the linear combination of vectors and the dot product. This algebraic approach combined with a multilevel MPI+OpenMP+OpenCL parallelization naturally provides portability. The performance has been studied on different architectures including multicore CPUs, Intel Xeon Phi accelerators and GPUs of AMD and NVIDIA. The multi-GPU scalability is demonstrated up to 256 devices. The heterogeneous execution is tested on a CPU+GPU hybrid cluster. Finally, results of the direct numerical simulation of a turbulent flow in a 3D air-filled differentially heated cavity are presented to show the capabilities of the HPC^2 dealing with large-scale CFD simulations.

1. Introduction

Massively-parallel devices of various architectures are being adopted by the newest supercomputers in order to overcome the actual power constraint in the context of the exascale challenge [1]. This trend is being reflected in most of the fields that rely on large-scale simulations such as computational fluid dynamics (CFD). Examples of CFD applications using accelerators can be found, for instance, in [2] (single-GPU, portable, OpenCL), [3–5] (multi-GPU, vendor-locked, CUDA) and [6] (petascale, multi-GPU, portable, CUDA+OpenCL).

Although the majority of problems in the field of mathematical physics involve sparse matrix and vector operations and hence algorithms with very low arithmetic intensity, most of the emerging HPC architectures are FLOP-oriented, *i.e.* FLOPS to memory bandwidth ratio is very high. Consequently, the achievable performance is usually reduced to a small fraction of the peak performance as proven by the HPCG Benchmark [7] results.

Therefore, in the design of large-scale simulation tools, software portability and efficiency are of crucial importance. The computing operations that form the algorithm, so-called kernels, must be compatible with distributed- and shared-memory MIMD parallelism and, more importantly, with stream processing, which is a more restrictive parallel paradigm. Consequently, the fewer the kernels of an application, the easier it is to provide portability. Furthermore, if the majority of kernels represent linear algebra operations, then standard optimized libraries (*e.g.* ATLAS, clBLAST) or specialized in-house implementations can be used and easily switched.

In this context, we proposed in a previous work [8] a portable algebraic implementation approach for direct numerical simulations (DNS) and large eddy simulation (LES) of incompressible turbulent flows on unstructured meshes. Roughly, the implementation consists in replacing classical stencil data structures and sweeps by algebraic data structures and kernels. As a result, the algorithm relies on a reduced set of only three algebraic operations: the sparse matrix-vector product (*SpMV*), the linear combination of vectors (*axpy*) and the dot product (*dot*).

On the other hand, the hybridization of HPC systems makes the design of simulation codes a rather complex problem. Heterogeneous implementations such as an MPI+OpenMP+OpenCL parallelization [9] can target a wide range of architectures and combine different kinds of parallelism. Hence, they are becoming increasingly necessary in order to engage all available computing power and memory throughput of CPUs and accelerators. Examples of CFD codes capable of heterogeneous computing can be found, for

* Corresponding author.
E-mail addresses: xavier@cttc.upc.edu (X. Álvarez), andrey@cttc.upc.edu (A. Gorobets), xavi@cttc.upc.edu (F.X. Trias).

https://doi.org/10.1016/j.compfluid.2018.01.034
0045-7930/© 2018 Published by Elsevier Ltd.

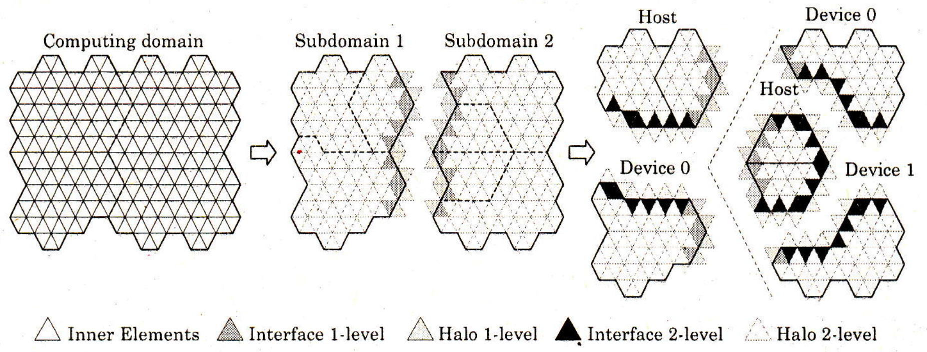


Fig. 1. Multi-level decomposition example for a cell-centred scheme among two dual-CPU nodes with one and two acceleration devices, respectively.

instance, in [10] the performance of the PyFR framework is tested on a hybrid node with a multicore CPU, NVIDIA and AMD GPUs. Further, in [11] the scalability of the HOSTA code is tested on up to 1024 TianHe-1A hybrid nodes.

Following the spirit of Oyarzun et al. [8], and increasing the level of abstraction, we present in this paper the HPC^2 (Heterogeneous Portable Code for HPC). It is a fully-portable, algebra-based framework capable of heterogeneous computing with many potential applications in the fields of computational physics and mathematics. This framework aims to provide a user-friendly environment suitable for writing numerical algorithms in terms of portable linear algebra kernels.

2. Multi-level domain decomposition

The computational domain is essentially a graph, *i.e.* a set of objects in which some pairs are in some sense related (such as mesh nodes, cells, faces, vertices, etc.), which may be subject to calculations. It typically arises from the spatial discretization of the physical domain, forming a fundamental part of many applications. The optimal distribution of the workload of the computational domain across the HPC system is of great importance in heterogeneous computing for attaining maximum performance.

By way of example, let us consider a generic numerical algorithm which operates on a computational domain. The algorithm is to be executed on an HPC system that consists of computing nodes interconnected via a communication infrastructure. Hence, a traditional first-level domain decomposition approach with MPI parallelization is used in order to distribute the workload among multiple nodes. In doing so, domain elements are assigned to subdomains using a partitioning library (*e.g.* ParMETIS [12]) that fulfils the requested load balancing and minimizes the number of couplings between cells of different subdomains. As a result, first-level subdomain elements are classified into *Inner* and *Interface* categories (see Fig. 1). Namely, *Interface* elements are those coupled with elements from other subdomains. Consequently, those other's neighbouring elements form a *Halo*. A communication between parallel processes is needed in order to update the data in *Halo* elements needed by a kernel when processing *Interface* elements. The *Halo* update is represented with non-blocking point-to-point MPI communications between neighbouring processes. An overlap of these communications with computations partially eliminates the data transfer overhead. In such a case, the *Halo* update is carried out simultaneously to the execution of the kernel only for the *Inner* elements. Afterwards, once the *Halo* is updated, the *Interface* elements are computed.

Similarly, first-level subdomains are decomposed further in order to distribute the workload of each node among its computing devices, such as multiple CPUs (called host) and co-processors of different kinds (called devices), as shown in the right part of Fig. 1.

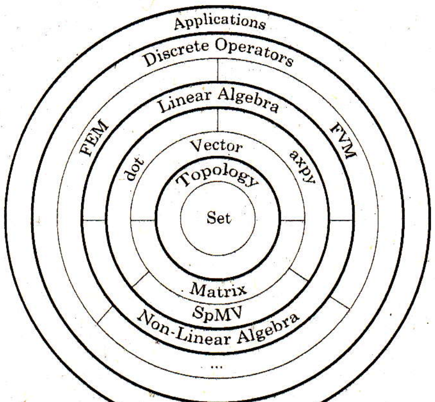


Fig. 2. Representation of the HPC^2 code structure.

This second-level decomposition must conform to the actual performance of devices for the sake of load balancing. As a result, at the second level, the *Halo* and *Interface* elements are classified as (1) external ones that need MPI communications because are coupled with other subdomains of the first-level decomposition, and (2) internal ones that only participate in the intra-node exchanges. The external *Interface* and *Halo* elements which are assigned to a device with a separate memory space need more expensive multi-stage device-host-MPI-host-device communications. The volume of this expensive *Interface* is reduced several times with the two-level partitioning since in the one-level case the internal *Interface* would become external.

Finally, the third-level decomposition among NUMA nodes of the host (*e.g.* CPU sockets or parts of a multicore CPU grouped in a shared L3 inner cache ring) allows allocating data in accordance with the physical resources to which a group of threads is assigned.

3. The HPC^2 framework

In this section, we present the HPC^2: a fully-portable, algebra-based framework suitable for heterogeneous computing with the aim of providing a user-friendly environment for writing algorithms in the fields of computational physics and mathematics.

3.1. Structure of HPC^2

The code is structured following a multi-layer design represented in Fig. 2 as concentric rings. In this scheme, the layers are defined to be detached maintaining the object dependency restricted to the inner layers. Therefore, each outer layer represents a higher level of abstraction.

The first layer (centre) is composed of the *Set* and *Topology* objects. This layer represents the computational domain (detailed in Section 2) hence it is the core of any numerical simulation. Firstly, the *Set* is a basic data structure which aims to mimic the algebraic concept of a set *i.e.* a collection of objects of some kind. It is designed to be automatically distributed in the system and assigned to devices at runtime according to the execution parameters. Thus, it becomes generic and architecture-independent from the outer layers' point of view. Secondly, the *Topology* is designed to be in agreement with a *Set*. It consists of the representation of the couplings between the objects of the *Set*. Therefore, it contains the required information to perform the data exchanges (*Halo* updates). Note that the *Topology* is bounded to a *Set*, then it can only manage the data exchanges of objects belonging to that *Set*. However, different *Topology* may be assigned to the same *Set* depending on the numerical schemes (*e.g.* the second- and fourth-order schemes define different couplings between the same set of unknowns).

In the second layer, two more complex algebraic objects, the *Vector* and the *Matrix*, are derived from *Set*. The *Vector* object represents discrete functions on the computational domain (*e.g.* pressure, velocity, temperature). To comply with the algebraic concept of vector it must be equipped with the algebraic operations: dot product, scalar multiplication, vector addition, and linear combination. These operations are contained in the *dot* and *axpy* kernels. The *Matrix* object is provided with the matrix multiplication, represented by the *SpMV* kernel, in order to perform linear transformations.

The third layer contains linear and non-linear algorithms. These algorithms are written using only *Matrix* and*Vector* kernels, and hence they only maintain an inner-layer dependency. Some examples of algorithms already implemented are the Conjugate Gradient. solver, the Adams-Bashforth time integrator and the Courant–Friedrichs–Lewy (CFL) condition.

The fourth layer consists of the preprocessing mechanisms which can be defined for different numerical methods. Namely, given a mesh, the preprocessing constructs the *Set* and *Topology* objects. Then, it generates the coefficients of the operators such as Gradient, Divergence or Laplacian. Additionally, this layer can involve external simulation codes which generate the operators as an input for the HPC^2 time integration core.

Finally, the outermost layer is left for applications. The preprocessing mechanisms are used to generate the required *Matrix* and *Vector* objects. The combination of these objects, and its kernels, together with the algorithms described in the third layer allows the implementation of complex algorithms in the fields of computational physics and mathematics. By way of example, the reader is referred to the Algorithm 1 (in Section 4) for modelling DNS of incompressible turbulent flows with heat transfer is composed of only three linear algebra kernels: *SpMV*, *dot*, *axpy*.

3.2. Software implementation details

The structure of HPC^2 aims to restrict the implementation specifics to the inner layers, *i.e.* the core and algebraic layers. Our heterogeneous implementation relies on MPI, OpenMP and OpenCL frameworks. Firstly, the non-blocking MPI point-to-point communications are used for distributed-memory parallelization. Secondly, OpenMP is used for shared-memory MIMD parallelization for multicore CPUs and manycore accelerators. The dynamic loop scheduling is mostly used in order to avoid imbalance between threads that may appear due to the interference with OpenCL and MPI processes. Additionally, vectorization for SIMD extensions on the lowest level is achieved with compiler-specific directives,

Algorithm 1 Time-integration step.

```
1. Compute the convective, the diffusiveand the source term of momentum Eq. (1):
   $\mathrm{R}(\boldsymbol{u}_s^n, \boldsymbol{u}_c^n, \boldsymbol{\theta}_c^n) \equiv -\mathrm{C}_c^{3d}(\boldsymbol{u}_s^n)\boldsymbol{u}_c^n - \mathrm{D}_c^{3d}\boldsymbol{u}_c^n + \boldsymbol{f}_c(\boldsymbol{\theta}_c^n)$
2. Compute the predictor velocity: $\boldsymbol{u}_c^p = \boldsymbol{u}_c^n + \Delta t\left\{\frac{3}{2}\mathrm{R}(\boldsymbol{u}_s^n, \boldsymbol{u}_c^n) - \frac{1}{2}\mathrm{R}\left(\boldsymbol{u}_s^{n-1}, \boldsymbol{u}_c^{n-1}\right)\right\}$
3. Solve the Poisson equation given in Eq. (4): $\mathrm{L}\tilde{\boldsymbol{p}}_c^{n+1} = \mathrm{M}\boldsymbol{u}_s^p$ where
   $\boldsymbol{u}_s^p = \Gamma_{c\to s}\boldsymbol{u}_c^p$
4. Correct the staggered velocity field: $\boldsymbol{u}_s^{n+1} = \boldsymbol{u}_s^p - \mathrm{G}\tilde{\boldsymbol{p}}_c^{n+1}$ where
   $\mathrm{G} = -\Omega_s^{-1}\mathrm{M}^T$
5. Correct the cell centered velocity field: $\boldsymbol{u}_c^{n+1} = \boldsymbol{u}_c^p - \mathrm{G}_c\tilde{\boldsymbol{p}}_c^{n+1}$
   where
   $\mathrm{G}_c \doteq -\Gamma_{s\to c}\Omega_s^{-1}\mathrm{M}^T$
6. Compute the convective and the diffusive terms intemperature transport Eq. (2):
   $\mathrm{R}_\theta\left(\boldsymbol{u}_s^n, \boldsymbol{\theta}_c^n\right) \equiv -\mathrm{C}_c(\boldsymbol{u}_s^n)\boldsymbol{\theta}_c^n - \mathrm{Pr}^{-1}\,\mathrm{D}_c\boldsymbol{\theta}_c^n$
7. Compute temperature at the next time-step:
   $\boldsymbol{\theta}_c^{n+1} = \boldsymbol{\theta}_c^n + \Delta t\left\{\frac{3}{2}\mathrm{R}_\theta\left(\boldsymbol{u}_s^n, \boldsymbol{\theta}_c^n\right) - \frac{1}{2}\mathrm{R}_\theta\left(\boldsymbol{u}_s^{n-1}n, \boldsymbol{\theta}_c^{n-1}\right)\right\}$
```

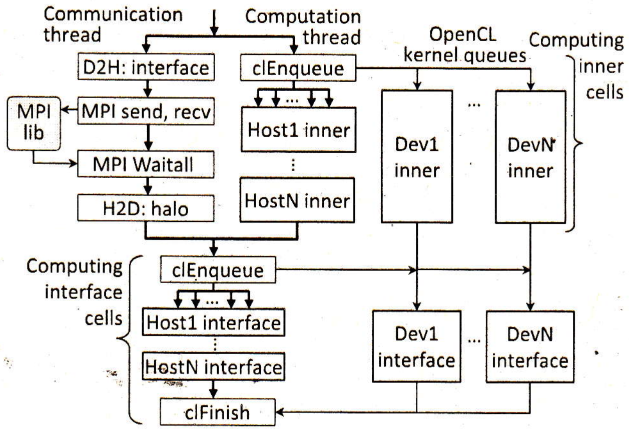


Fig. 3. Heterogeneous execution of a kernel over a multi-level decomposed domain.

such as *#pragma ivdep*for the Intel compiler. Finally, OpenCL implementation provides the kernels portability across various stream processing-based accelerators, including AMD and NVIDIA GPUs, FPGA accelerators and ARM-based systems-on-a-chip (SoCs).

The heterogeneous execution mode is implemented using nested OpenMP regions (Fig. 3). The outer parallel region spawns two threads: one for handling communications and another for computations. The communication thread executes device-to-host (D2H), MPI, and host-to-device (H2D) transfers. The computing thread submits kernels for the background OpenCL execution, then OpenMP-parallel processing is carried out within the inner parallel region. In doing so, the OpenMP and OpenCL computations are carried out simultaneously with the data exchanges engaging all available computing resources and hiding the communication overhead.

In order to optimize the matrix data structures and kernels, it is necessary to reorder the rows and provide it with a proper storage format depending on its sparsity pattern and the target architecture. The reordering aims to improve the data locality by reducing the matrix bandwidth [13]. The implemented storage formats include the standard Compressed Sparse Row (CSR) and different variants of the ELLPACK (sliced and block-transposed). Further details on the implementation and the performance of these formats can be found in our previous work [8]. Note, this reordering and

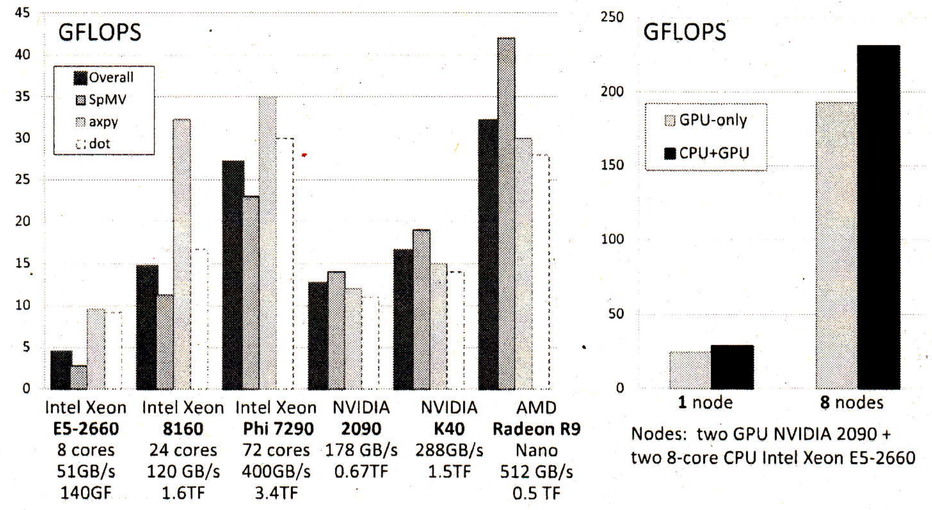


Fig. 4. Left: performance of the overall DNS algorithm (Algorithm 1 and the three basic kernels tested on different devices. Right: heterogeneous execution of the overall DNS algorithm vs GPU-only.

storage adaptation are internal routines that are hidden from the outer layers.

3.3. Performance study

Firstly, the performance of HPC^2 depends mainly on the algebraic kernels that compose its core. It must be noted that these algebraic operations have very low arithmetic intensity. For double-precision values, the FLOP per byte ratio is typically around 1/8 (one operation per 8-byte argument). Therefore, it is clearly a memory-bounded application that requires a lot of attention to memory access optimization. For this reason, the theoretically achievable performance can hardly reach several few percents of the device's theoretical peak. For instance, for NVIDIA 2090 GPU this limit is around 3% (*i.e.* $(0.125FLOP/Byte \cdot 178GB/s)/670GFLOPS$). This performance level is rather consistent with the results of the well-known HPCG benchmark [7] that reproduces a memory-bounded sparse algebraic application.

Single device tests have been run to study the performance of the HPC^2 on the following devices: Intel Xeon E5-2660, Intel Xeon E5-2620 v2, Intel Xeon 8160, Intel Xeon Phi 7290, NVIDIA Tesla 2090, NVIDIA Tesla K40, and AMD Radeon R9 Nano. Single-device results shown in Fig. 4 (left) illustrate the performance comparison for the overall DNS algorithm (see Algorithm 1 in Section 4) and its three major kernels on different kinds of devices: several generations of multicore and manycore CPUs and GPUs of AMD and NVIDIA. Additionally, performance on ARM-based SoCs can be found in our previous work [14]. The mesh size per device was around 1 million cells unstructured and hexa-dominant). As expected, the achieved performance is directly related to the bandwidth capacity of the devices. Consequently, the AMD GPU outperforms the FLOPS-oriented high-end computing devices due to its higher memory bandwidth. On the other hand, the attained performance and its ratio between devices differ for each kernel. This requires a careful workload balancing based on the performance of the overall algorithm but not the separate kernels.

The benefits of the heterogeneous CPU+GPU execution have been measured on a hybrid cluster using two 8-core CPUs (E5-2660) and two GPUs (NVIDIA 2090). Comparison with the GPU-only execution in shown in Fig. 4 (right) for an unstructured hexa-dominant mesh with 1 million cells per node. The performance gain compared to the GPU-only mode was 19%. Furthermore, the heterogeneous efficiency (*i.e.* the ratio between the heterogeneous performance and the sum of the CPU-only and the GPU-only performance) appeared to be near 100% on 8 nodes. This efficiency was expected to reduce since the CPUs should be more involved in communications. However, the communication overhead appears to be much more efficiently hidden when overlapping with GPUs.

Finally, multi-GPU strong and weak scaling results are shown in Fig. 5 for hexahedral meshes that represent the computational domain of the DNS configuration described in the following section. The HPC5 supercomputer of the Kurchatov Institute was used for these tests. Its nodes are equipped with two dual-GPU NVIDIA K80 devices. It can be observed that the parallel efficiency goes down rather rapidly in the strong speedup tests, allowing to speed-up around 8 × at a reasonable efficiency level. This is due to the natural fact that the GPUs decrease throughput notably when the workload per device goes down. In contrast, the weak scaling efficiency with a sufficient workload per device appears rather high. For the mesh of 1.3 billion cells, it is 94% on 256 GPUs when scaling from one 4-GPU node to 64 nodes with the load of 5 million cells per device. At the smaller workload of 1 million cells per device it is lower, 67% (around 1.5 times slowdown), when scaling from 1 to 256 devices because, firstly, the computing load is not sufficient to hide all the communications, and, secondly, the scaling range is 4 times bigger.

4. Challenging HPC^2: DNS of a turbulent differentially heated cavity

A DNS of a turbulent air-filled differentially heated cavity (DHC) has been chosen as a first CFD case to show the capabilities of the HPC^2 dealing with large-scale CFD simulations. Firstly, the description of the case in conjunction with the numerical methods is detailed. Then, the DHC results are briefly presented.

4.1. Mathematical model and numerical method

We consider a cavity of height H, width L and depth D filled with an incompressible Newtonian viscous fluid of kinematic viscosity ν, thermal diffusivity α and density ρ. The geometry of the problem is displayed in Fig. 6(left). The Boussinesq approximation is used to account for the density variations. Thermal radiation is neglected. Under these assumptions, the velocity, $\boldsymbol{u}$, and the temperature, θ, are governed by the following set of dimensionless

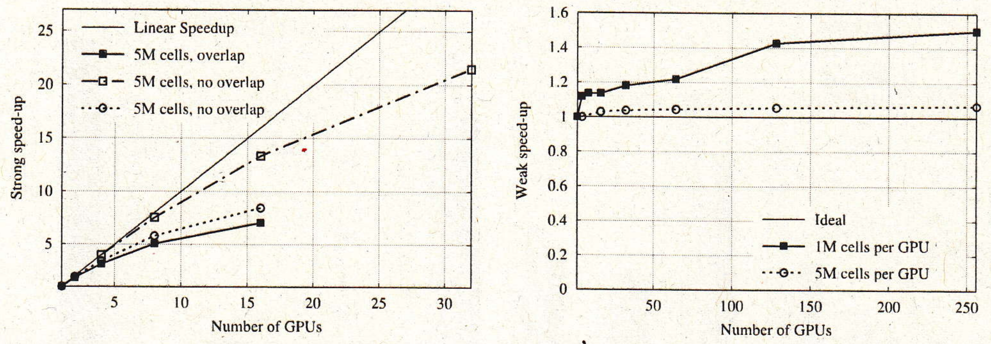


Fig. 5. Strong (left) and weak (right) scaling on multiple GPUs for different mesh sizes.

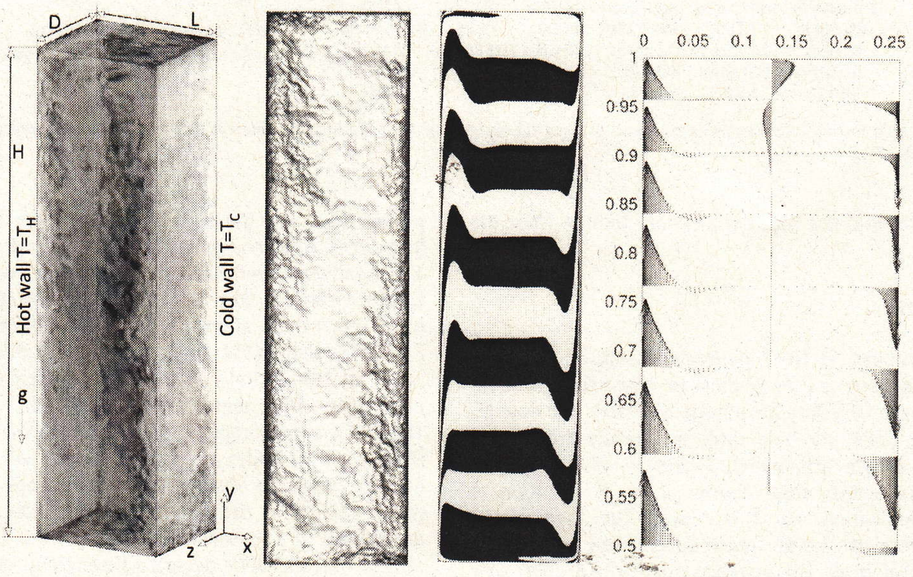


Fig. 6. From left to right: DHC schema, instantaneous schlieren-like snapshot from the DNS and the averaged temperature field at the cavity mid-depth (the isotherms are uniformly distributed between −0.5 and 0.5), the airflow map in the upper part of the cavity.

PDEs

$$\partial_t \boldsymbol{u} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u} = PrRa^{-1/2}\nabla^2\boldsymbol{u} - \nabla p + \boldsymbol{f}, \tag{1}$$

$$\partial_t \theta + (\boldsymbol{u}\cdot\nabla)\theta = Ra^{-1/2}\nabla^2\theta, \tag{2}$$

where $\Pr = \nu/\alpha$, $Ra = (g\beta\Delta\theta H^3)/(\nu\alpha)$ and $\boldsymbol{f} = (0, \Pr\theta, 0)$ (Boussinesq approximation) are the Prandtl and Rayleigh number (based on the cavity height), and the body force vector, respectively. Notice that with the reference quantities, $L_{ref} = H$ and $t_{ref} = (H^2/\alpha)Ra^{-1/2}$, the vertical buoyant velocity, $\Pr^{1/2}$, and the characteristic dimensionless Brunt-Väisälä frequency, N, are independent of the Ra. The configuration considered here resembles the experimental set-up performed by Saury et al. [15] and Belleoud et al. [16]: the height, H/L, and depth, D/L, aspect ratios are 3.84 and 0.86, whereas the Rayleigh and Prandtl numbers are $Ra = 1.2\times10^{11}$ and $\Pr = 0.71$ (air), respectively. The cavity is subjected to a temperature difference $\Delta\theta$ across the vertical isothermal walls ($\theta(0, y, z) = 0.5$, $\theta(L/H, y, z) = -0.5$). The temperature at the rest of walls is given by the "Fully Realistic" boundary conditions proposed in [17]. They are time-independent analytical functions that fit the experimental data of Salat et al. [18]. The no-slip boundary condition is imposed on the walls.

The governing Eqs. (1) and (2) are discretized using a symmetry-preserving discretization [19]. Shortly, the temporal evolution of the spatially discrete velocity vector, $\boldsymbol{u}_c$, is governed by the following operator-based finite-volume discretization of Eqs. (1)

$$\Omega_c^{3d}\frac{d\boldsymbol{u}_c}{dt} + \mathsf{C}_c^{3d}(\boldsymbol{u}_s)\boldsymbol{u}_c + \mathsf{D}_c^{3d}\boldsymbol{u}_c + \Omega_c^{3d}\mathsf{G}_c\boldsymbol{p}_c = \boldsymbol{f}_c, \quad \mathsf{M}\boldsymbol{u}_s = \boldsymbol{0}_c,$$

where the $\boldsymbol{p}_c \in \mathbb{R}^n$ and $\boldsymbol{u}_c \in \mathbb{R}^{3n}$ are the cell-centred pressure and velocity fields. For simplicity, $\boldsymbol{u}_c$ is defined as a column vector and arranged as $\boldsymbol{u}_c = (\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3)^T$, where $\boldsymbol{u}_i = ((u_i)_1, (u_i)_2, \ldots, (u_i)_n)^T$ are the vectors containing the velocity components corresponding to the x_i-spatial direction. The auxiliary discrete staggered velocity $\boldsymbol{u}_s = ((u_s)_1, (u_s)_2, \ldots, (u_s)_m)^T \in \mathbb{R}^m$ is related to the centered velocity field via a linear shift transformation (interpolation) $\Gamma_{c\to s} \in \mathbb{R}^{m\times 3n}$, $\boldsymbol{u}_s \equiv \Gamma_{c\to s}\boldsymbol{u}_c$. The dimensions of these vectors, n and m, are the number of control volumes and faces on the computational domain, respectively. The subindices c and s refer to whether the variables are cell-centered or staggered at the faces. The matrices $\Omega_c^{3d} \in \mathbb{R}^{3n\times 3n}$, $\mathsf{C}_c^{3d}(\boldsymbol{u}_s) \in \mathbb{R}^{3n\times 3n}$ and $\mathsf{D}_c^{3d} \in \mathbb{R}^{3n\times 3n}$ are block diagonal matrices given by

$$\Omega_c^{3d} = \mathsf{I}_3 \otimes \Omega_c, \qquad \mathsf{C}_c^{3d}(\boldsymbol{u}_s) = \mathsf{I}_3 \otimes \mathsf{C}_c(\boldsymbol{u}_s), \qquad \mathsf{D}_c^{3d} = \mathsf{I}_3 \otimes \mathsf{D}_c,$$

where $\mathsf{I}_3 \in \mathbb{R}^{3\times 3}$ is the identity matrix. $\mathsf{C}_c(\boldsymbol{u}_s) \in \mathbb{R}^{n\times n}$ and $\mathsf{D}_c \in \mathbb{R}^{n\times n}$ are the collocated convective and diffusive operators, respectively. The temporal evolution of the discrete temperature $\boldsymbol{\theta}_c \in \mathbb{R}^n$ (see Eq. (2)) is discretized in the same vein. For a detailed explanation of the spatial discretization, the reader is referred to

Table 1
Physical and numerical simulation parameters of the DNS of the turbulent DHC displayed in Fig. 6. From left to right: number of control volumes and concentration factors for each spatial direction, the size of the first off-wall control volume in the x-direction (also in wall-units), the non-dimensional time-step, the starting time for averaging and the time-integration period.

N_x	N_y	N_z	γ_x	γ_y	γ_z	$(\Delta x)_{min}$	$(\Delta x)^+_{min}$	Δt	t_0	Δt_{avg}
450	900	256	2	1	1	4.28×10^{-5}	≤ 0.5	3.65×10^{-4}	≈ 300	≈ 300

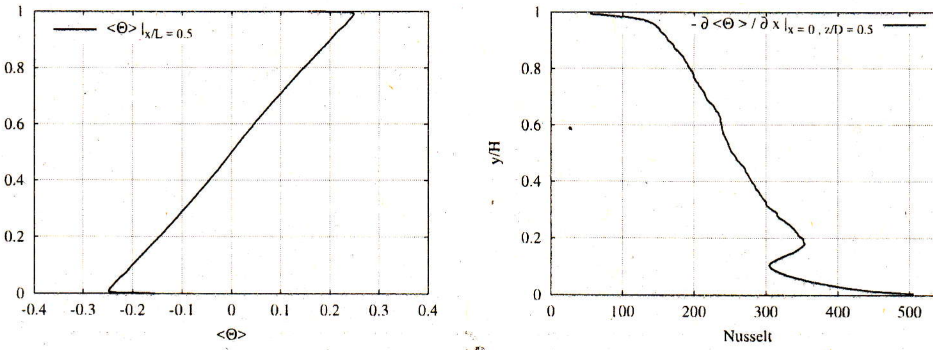


Fig. 7. Left: average temperature profiles at the cavity mid-depth at mid-width. Right: averaged Nusselt number at the cavity mid-depth.

Trias et al. [19]. Regarding the temporal discretization, a second-order explicit one-leg scheme is used for both the convective and the diffusive terms [20]. Finally, the pressure-velocity coupling is solved by means of a classical fractional step projection method [21]: a predictor velocity, $\boldsymbol{u}_s^p$, is explicitly evaluated without considering the contribution of the pressure gradient. Then, by imposing the incompressibility constraint, $\mathsf{M}\boldsymbol{u}_s^{n+1} = \boldsymbol{0}_c$, it leads to a Poisson equation for $\tilde{\boldsymbol{p}}_c^{n+1}$ to be solved once each time-step,

$$\mathsf{L}\tilde{\boldsymbol{p}}_c^{n+1} = \mathsf{M}\boldsymbol{u}_s^p \quad \text{with} \quad \mathsf{L} = -\mathsf{M}\Omega_s^{-1}\mathsf{M}^T, \tag{3}$$

where $\tilde{\boldsymbol{p}}_c = \Delta t \boldsymbol{p}_c$ and the discrete Laplacian operator, L, is represented by a symmetric negative semi-definite matrix.

In summary, the method is based on only five basic (linear) operators: the cell-centered and staggered control volumes, Ω_c and Ω_s, the matrix containing the face normal vectors, $\mathtt{N_s}$, the cell-to-face scalar field interpolation, $\Pi_{c\to s}$ and the divergence operator, M. Once these operators are constructed, the rest follows straightforwardly from them. The algorithm to solve one time-integration step in outlined in Algorithm 1. At this point, it must be noted that, except the non-linear convective term, $\mathsf{C}_c^{3d}(\boldsymbol{u}_s^n)\boldsymbol{u}_c^n$ and $\mathsf{C}_c(\boldsymbol{u}_s^n)\boldsymbol{\theta}_c^n$, all the operations correspond to sparse matrix-vector products (*SpMV*), most of the them sharing the same matrix portrait. Regarding the convection (steps 1 and 2 in Algorithm 1), it can also be reduced to *SpMV* operations by simply noticing that the coefficients of the convective operator, $\mathsf{C}_c(\boldsymbol{u}_s^n)$, must be re-computed accordingly to the adopted numerical schemes [19]. Moreover, the computation of these coefficients can also be viewed as a *SpMV*. Therefore, the convective operator is represented as a concatenation of two *SpMVs*: (i) firstly, to compute the coefficients of the convective operator, $\mathsf{C}_c(\boldsymbol{u}_s^n)$, (ii) then, to compute the matrix-vector product to obtain the resulting vector, *e.g.* $\mathsf{C}_c(\boldsymbol{u}_s^n)\boldsymbol{\theta}_c^n$.

Regarding the time-integration scheme (steps 2 and 7 in Algorithm 1), and without loss of generality, a second-order Adams-Bashforth has been adopted here. Since it is a fully explicit schemes a CFL-like condition is required in order to keep the numerical scheme inside the stability region [20]. This necessarily leads to rather small time-steps, Δt and subsequently to a good initial guess for the Poisson equation. This justifies the fact that a relatively simple linear solver for the Poisson equation (step 3 in Algorithm 1) suffices to maintain the norm of the divergence of the velocity field, $\mathsf{M}\boldsymbol{u}_s^{n+1}$, at a low enough level [22]. Furthermore, since the matrix, −L, is symmetric and positive-definite, a Preconditioned Conjugate Gradient is used with a simple *SpMV*-based preconditioner (either the Jacobi or the approximate inverse). In conclusion, the overall Algorithm 1 relies on the set of three basic algebra operations: *SpMV*, *dot* and *axpy*.

4.2. Results and discussion

Since no subgrid-scale model is used in the computation, the grid resolution and the time-step, Δt, have to be fine enough to resolve all the relevant turbulence scales. Furthermore, the starting time for averaging, t_0, and the time-integration period, Δt_{avg}, must also be long enough to properly evaluate the flow statistics. The procedure followed to verify the simulation results is analogous to our previous DNS work [23,24]. In this case, the averages over the three statistically invariant transformations (time, mid-depth plane and central point symmetry) have been carried out for all fields and the grid points in the three wall-normal directions are distributed using a hyperbolic-tangent function, *i.e.* for the *x*-direction $x_i = \frac{1}{2}(1 + \frac{\tanh\{\gamma_x(2(i-1)/N_x-1)\}}{\tanh\gamma_x})$. For details about the physical and numerical parameters see Table 1. Hereafter, the angular brackets operator $\langle\cdot\rangle$ denotes averaged variables.

Instantaneous flow fields displayed in Fig. 6 illustrate the inherent flow complexity of this configuration. Namely, the vertical boundary layers remain laminar only in their upstream part up to the point where the waves traveling downstream grow up enough to disrupt the boundary layers ejecting large unsteady eddies. An accurate prediction of the flow structure in the cavity lies on the ability to correctly locate the transition to turbulence while the high sensitivity of the thermal boundary layer to external disturbances makes it difficult to predict (see results for a similar DHC configuration in [24,25], for instance). In this case, the transition occurs around $y \approx 0.2$ (see the peak of the averaged local Nusselt number displayed in the right part of Fig. 6).

The average temperature field and the airflow map are displayed in Fig. 6 (right). The cavity is almost uniformly stratified with a dimensionless stratification of $S \approx 0.45$ (see Fig. 7, left). This value is in a rather good agreement with the experimental results obtained by Saury et al. [15] ($S = 0.44 \pm 0.03$ with $\epsilon = 0.1$ and $S = 0.57 \pm 0.03$ with $\epsilon = 0.6$, where ϵ is the wall emissivity). The averaged Nusselt number at the cavity mid-depth is displayed in Fig. 7 (right). The profile is again rather similar to the experimen-

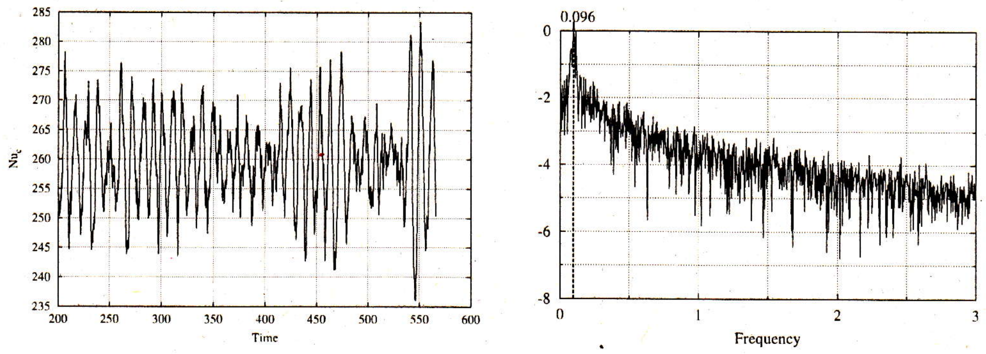


Fig. 8. Time evolution of the Nusselt number at the vertical mid-plane (left) and its normalized density power spectrum (right).

tal results obtained by Saury et al. [15]. In this case, the transition point occurs at a slightly more upstream position. The peak of the averaged local Nusselt number is located at $y \approx 0.2$ whereas in the experimental results this point is at $y \approx 0.3$. Integrating the averaged local Nusselt number over the y-direction, the overall Nusselt is determined. In this case, $\langle Nu \rangle = 259.2$, a slightly higher value that the one obtained by D. Saury et al. [15], *i.e.* $\langle Nu \rangle = 231 \pm 30$ but very similar to the value obtained by means of LES, *i.e.* $\langle Nu \rangle = 254$ (see Fig. 11 in [15]).

Another important feature of this kind of configuration is the presence of internal waves. Although in the cavity core the averaged velocity (and its fluctuations) are much smaller compared with those observed in the vertical boundary layers, simulations show that in this region isotherms oscillate around the mean horizontal profile. As mentioned-above, the cavity core remains well stratified (see Figs. 6 and 7, left); therefore, this phenomenon can be attributed to internal waves. This can be confirmed by analysing the Nusselt number through the vertical mid-plane, Nu_c. The time evolution and the normalized density power spectrum are respectively displayed in Figs. 8. The peak in the spectrum is located at 0.096 which is in a good agreement with the dimensionless Brunt-Väisälä frequency, $N = (S\,\mathrm{Pr})^{0.5}/(2\pi)$, where S is the dimensionless stratification of the time-averaged temperature, *i.e.* $N \approx 0.09$. Both values are very similar confirming that internal waves are permanently excited by the eddies ejected from the vertical boundary layer. Detailed results including turbulent statistics can be downloaded in the following link [26].

5. Conclusions

Motivated by the constant evolution of HPC architectures, the aim of this paper was to design a fully-portable, algebra-based framework suitable for heterogeneous computing with the aim of providing a user-friendly environment for writing algorithms in the fields of computational physics and mathematics. As a computing novelty, the heterogeneous MPI+OpenMP+OpenCL implementation of kernels has been combined with a multi-level domain decomposition strategy for distributing the workload among heterogeneous computing resources. Results have shown that the heterogeneous performance of the HPC^2 on a hybrid CPU+GPU cluster is nearly identical to the sum of the CPU-only and the GPU-only performance. The multi-GPU scalability of a CFD simulation has been demonstrated on up to 64 nodes equipped with 4 GPU devices. In addition, the performance has been studied on various architectures including different generations of multicore-CPUs, AMD and NVIDIA GPUs, manycore accelerators (with the same kernel code, only changing the local workgroup sizes). These results demonstrate the portability of the proposed approach.

Acknowledgments

The work has been financially supported by the Ministerio de Economía y Competitividad, Spain (ENE2014-60577-R). X. Á. is supported by a *FI* predoctoral contract (FI_B-2017-00614). A. G. is supported by the Russian Science Foundation (grant 15-11-30039). F. X. T. is supported by a *Ramón y Cajal* postdoctoral contract (RYC-2012-11996). R. B. is supported by a *Juan de la Cierva* posdoctoral grant (IJCI-2014-21034). This work has been carried out using computing resources of the federal collective usage center Complex for Simulation and Data Processing for Mega-science Facilities at NRC Kurchatov Institute, http://ckp.nrcki.ru/; the Barcelona Supercomputing Center; the Center for collective use of HPC computing resources at Lomonosov Moscow State University; the Joint Supercomputer Center of the Russian Academy of Sciences; the KIAM RAS. The authors thankfully acknowledge these institutions.

References

[1] Dongarra J, et al. The international exascale software project roadmap. Int J High Perform Comput Appl 2011;25(1):3–60.
[2] Rossi R, Mossaiby F, Idelsohn SR. A portable openCL-based unstructured edge-based finite element Navier–Stokes solver on graphics hardware. Comput Fluids 2013;81:134–44.
[3] Jacobsen DA, Senocak I. Multi-level parallelism for incompressible flow computations on GPU clusters. Parallel Comput 2013;39(1):1–20.
[4] Khajeh-Saeed A, Perot JB. Direct numerical simulation of turbulence using GPU accelerated supercomputers. J Comput Phys 2013;235:241–57.
[5] Zaspel P, Griebel M. Solving incompressible two-phase flows on multi-GPU clusters. Comput Fluids 2013;80(1):356–64.
[6] Vincent P, Witherden FD, Vermeire B, Park JS, Iyer A. Towards green aviation with python at petascale. In: International conference for high performance computing, networking, storage and analysis, SC; November 2017. p. 1–11.
[7] Dongarra J, Heroux M. HPCG benchmark: a new metric for ranking high performance computing systems. Technical Report June; 2013.
[8] Oyarzun G, Borrell R, Gorobets A, Oliva A. Portable implementation model for CFD simulations. Application to hybrid CPU/GPU supercomputers. Int. J. Comput. Fluid Dyn. 2017;31(9):396–411.
[9] Gorobets A, Trias FX, Oliva A. A parallel MPI+openMP+openCL algorithm for hybrid supercomputations of incompressible flows. Comput. Fluids 2013;88:764–72.
[10] Witherden FD, Vermeire B, Vincent P. Heterogeneous computing on mixed unstructured grids with pyFR. Comput Fluids 2015;120:173–86.
[11] Xu C, Deng X, Zhang L, Fang J, Wang G, Jiang Y, Cao W, Che Y, Wang Y, Wang Z, Liu W, Cheng X. Collaborating CPU and GPU for large-scale high-order CFD simulations with complex grids on the tianhe-1a supercomputer. J Comput Phys 2014;278(1):275–97.
[12] LaSalle D, Karypis G. Multi-threaded graph partitioning. In: Proceedings - IEEE 27th international parallel and distributed processing symposium, IPDPS 2013; 2013. p. 225–36.
[13] Cuthill E, McKee J. Reducing the bandwidth of sparse symmetric matrices. In: Proceedings of the 1969 24th national conference on -; 1969. p. 157–72.
[14] Oyarzun G, Borrell R, Gorobets A, Mantovani F, Oliva A. Efficient CFD code implementation for the ARM-based mont-blanc architecture. Future Gener Comput Syst 2018;79:786–96.
[15] Saury D, Rouger N, Djanna F, Penot F. Natural convection in an air-filled cavity: experimental results at large rayleigh numbers. Int Commun Heat Mass Transfer 2011;38:679–87.

[16] Belleoud P, Saury D, Lemonnier D. Coupled velocity and temperature measurements in an air-filled differentially heated cavity at ra=1.2e11. Int J Therm Sci 2018;123:151–61.

[17] Sergent A, Joubert P, Xin S, Le Quéré P. Resolving the stratification discrepancy of turbulent natural convection in differentially heated air-filled cavities. part II: end walls effects. Int J Heat Fluid Flow 2013;39:15–27.

[18] Salat J, Xin S, Joubert P, Sergent A, Penot F, Le Quéré P. Experimental and numerical investigation of turbulent natural convection in a large air-filled cavity. Int J Heat Fluid Flow 2004;25:824–32.

[19] Trias FX, Lehmkuhl O, Oliva A, Pérez-Segarra CD, Verstappen RWCP. Symmetry-preserving discretization of Navier–Stokes equations on collocated unstructured meshes. J Comput Phys 2014;258:246–67.

[20] Trias FX, Lehmkuhl O. A self-adaptive strategy for the time-integration of Navier–Stokes equations. Numer Heat Transfer Part B 2011;60(2):116–34.

[21] Chorin AJ. Numerical solution of the Navier–Stokes equations. Math Comput 1968;22:745–62.

[22] Gorobets A, Trias FX, Soria M, Oliva A. A scalable parallel poisson solver for three-dimensional problems with one periodic direction. Comput Fluids 2010;39:525–38.

[23] Trias FX, Gorobets A, Soria M, Oliva A. Direct numerical simulation of a differentially heated cavity of aspect ratio 4 with *ra*-number up to 10^{11} - part i: Numerical methods and time-averaged flow. Int J Heat Mass Transfer 2010;53:665–73.

[24] Trias FX, Gorobets A, Pérez-Segarra CD, Oliva A. DNS and regularization modeling of a turbulent differentially heated cavity of aspect ratio 5. Int J Heat Mass Transfer 2013;57:171–82.

[25] Barhaghi DG, Davidson L. Natural convection boundary layer in a 5:1 cavity. Phys Fluids 2007;19(12):125106.

[26] The DNS results presented in this paper are publicly available in http://www.cttc.upc.edu/downloads/DHC_Ra1_2e11.