Aquesta és una còpia de la versió author’s final draft d’un article publicat a la revista Computers & Fluids.

URL d’aquest document a UPCommons E-prints:

http://hdl.handle.net/2117/116810

Article publicat / Published paper:


© <2018>. Aquesta versió està disponible sota la llicència CC-BY-NC-ND 4.0 http://creativecommons.org/licenses/by-nc-nd/4.0/
HPC\textsuperscript{2}—A fully-portable, algebra-based framework for heterogeneous computing. Application to CFD

X. \textsuperscript{4}Álvarez\textsuperscript{a,b}, A. Gorobets\textsuperscript{a,c}, F.X. Trias\textsuperscript{a}, R. Borrell\textsuperscript{a}, C. Oyarzun\textsuperscript{a}

\textsuperscript{a} Heat and Mass Transfer Technological Center, Technical University of Catalonia, C/ Colom 11, Terrassa (Barcelona) 08222, Spain
\textsuperscript{b} Khalip Institute of Applied Mathematics RAS, Miasnitskaya Sq. 4, Moscow 125047, Russia
\textsuperscript{c} Termo Fluids, S.L.; C/ Magi Coler 8, Sabadell (Barcelona), 08204, Spain

\textbf{A R T I C L E   I N F O}

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

\textbf{A B S T R A C T}

The variety of computing architectures competing in the exascale race makes the portability of codes of major importance. In this work, the HPC\textsuperscript{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\textsuperscript{2} dealing with large-scale CFD simulations.

© 2018 Published by Elsevier Ltd.
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. [9], and increasing the level of abstraction, we present in this paper the HPC2 (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 fulfills 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. 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 HPC2 framework

In this section, we present the HPC2: 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 HPC2

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 (Hallo 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 HPCC2 time integration core.

Finally, the outermost layer is left for applications. The preprocessing mechanisms are needed 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 HPCC2 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 IMMD parallelization for multicore CPUs and mancore 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, such as #pragma ivdep for the Intel compiler. Finally, OpenCL implementation provides the kernels portability across various standard 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 [15]. Note, this reordering and
storage adaptation are internal routines that are hidden from the
outer layers.

3.3. Performance study

Firstly, the performance of HPC2 depends mainly on the alge-
braic kernels that compose its core. It must be noted that these al-
gebraic 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-bound application that requires a lot of attention to
memory access optimization. For this reason, the theoretically
achievable performance can hardly reach several few percent of
the device's theoretical peak. For instance, for NVIDIA 2090 GPU
this limit is around 3% (i.e. (0.125FLOP/byte · 178GB/s)/670GFLOPS).
This performance level is rather consistent with the results of
the well-known HPCG benchmark [?] that reproduces a memory-
bound sparse algebraic application.

Single device tests have been run to study the performance of
HPC2 on the following devices: Intel Xeon ES-2660, Intel Xeon
ES-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 compar-
ison 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 hex-dominant. As expected,
the achieved performance is directly related to the band-
width capacity of the devices. Consequently, the AMD GPU outper-
forms the FLOPS-oriented high-end computing devices due to its
higher memory bandwidth. On the other hand, the attained per-
formance 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 (ES-
2660) and two GPUs (NVIDIA 2090). Comparison with the GPU-
only execution is 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 per-
formance) 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 do-
main of the DNS configuration described in the following section.
The HPCS 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 natu-
ral fact that the GPUs decrease throughput notably when the work-
load 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 HPC2: DNS of a turbulent differently heated
cavity

A DNS of a turbulent air-filled differently heated cavity (DHC)
has been chosen as a first CFD case to show the capabilities of
the HPC2 dealing with large-scale CFD simulations. Firstly, the descrip-
tion of the case in conjunction with the numerical methods is de-
tailed. 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 vis-
cosity $\nu$, thermal diffusivity $\alpha$ and density $\rho$. The geometry of
the problem is displayed in Fig. 6 (left). The Boussinesq approxima-
tion is used to account for the density variations. Thermal radiation is
neglected. Under these assumptions, the velocity, $\mathbf{u}$, and the tem-
perature, $\theta$, are governed by the following set of dimensionless

\[ \begin{align*}
\nabla \cdot \mathbf{u} &= 0, \\
\rho \frac{D\mathbf{u}}{Dt} &= -\nabla P + \mu \nabla^2 \mathbf{u} + \rho g \mathbf{e}_z, \\
\rho \frac{D\theta}{Dt} &= \kappa \nabla^2 \theta + \eta \theta.
\end{align*} \]
The governing Eqs. (1) and (2) are discretized using a symmetry-preserving discretization [19]. Shortly, the temporal evolution of the spatially discrete velocity vector, $\mathbf{u}_i$, is governed by the following operator-based finite-volume discretization of Eqs. (1)

$$\frac{\partial^{2}}{\partial t^{2}}+C_{2}^{D}(u_{i})u_{i}+D_{2}^{D}u_{i}+\Delta^{2}C_{2}^{D}u_{i}=f_{c}$$

where the $p_{i} \in \mathbb{R}^{n}$ and $u_{i} \in \mathbb{R}^{3}$ are the cell-centred pressure and velocity fields, for simplicity, $u_{i}$ is defined as a column vector and arranged as $u_{i} = [(u_{i,1}), (u_{i,2}), \ldots, (u_{i,3})]^T$, where $u_{i} = [(u_{i,1}), (u_{i,2}), \ldots, (u_{i,3})]^T$ are the vectors containing the velocity components corresponding to the $X_i$-spatial direction. The auxiliary discrete staggered velocity $u_{i} = [(u_{i,1}), (u_{i,2}), \ldots, (u_{i,3})]^T \in \mathbb{R}^{m}$ is related to the centered velocity field via a linear shift transformation (interpolation) $\Gamma_{c-s} \in \mathbb{R}^{m \times 3}$, $u_{i} = \Gamma_{c-s}u_{c}$. 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 $C_{2}^{D} \in \mathbb{R}^{m \times 3}$, $D_{2}^{D} \in \mathbb{R}^{m \times 3}$ and $\Delta^{2}C_{2}^{D} \in \mathbb{R}^{m \times 3}$ are block diagonal matrices given by

$$\Delta^{2}C_{2}^{D} = I_{3} \otimes D_{c}, \quad C_{2}^{D}(u_{i}) = I_{3} \otimes C_{c}(u_{i}), \quad D_{2}^{D} = I_{3} \otimes D_{c}$$

where $I_{3} \in \mathbb{R}^{3 \times 3}$ is the identity matrix, $C_{c}(u_{i}) \in \mathbb{R}^{n \times n}$ and $D_{c} \in \mathbb{R}^{n \times n}$ are the colocated convective and diffusive operators, respectively. The temporal evolution of the discrete temperature $\theta_{c} \in \mathbb{R}^{n}$ (see Eq. (2)) is discretized in the same vein. For a detailed explanation of the spatial discretization, the reader is referred to the original paper.
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, \( \mathbf{u}_p^t \), is explicitly evaluated without considering the contribution of the pressure gradient. Then, by imposing the incompressibility constraint, \( \mathbf{u}_p^{t+1} = \mathbf{u}_c^t \), it leads to a Poisson equation for \( \mathbf{p}_c^{t+1} \) to be solved once each time-step,

\[
\mathbf{L} \mathbf{p}_c^{t+1} = \mathbf{u}_c^t \quad \text{with} \quad \mathbf{L} = - \mu \nabla^2 \mathbf{I} \tag{3}
\]

where \( \mathbf{p}_c = \Delta \mathbf{p}_c \) and the discrete Laplacian operator, \( \mathbf{L} \), 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, \( \Omega_c \) and \( \Omega_s \), the matrix containing the face normal vectors, \( \mathbf{n}_f \), the cell-to-face scalar field interpolation, \( \Pi_{f \leftarrow c} \), and the divergence operator, \( \nabla \cdot \). 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, \( \mathbf{C}^s(\mathbf{u}_f) \mathbf{u}_c^t \) and \( \mathbf{C}_c(\mathbf{u}_c^t) \mathbf{u}_c^t \), all the operators correspond to sparse matrix-vector products (SpMV), most of 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, \( \mathbf{C}_c(\mathbf{u}_c^t) \), 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, \( \mathbf{C}_c(\mathbf{u}_c^t) \); (ii) then, to compute the matrix-vector product to obtain the resulting vector, e.g., \( \mathbf{C}_c(\mathbf{u}_c^t) \mathbf{u}_c^t \).

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 schemes inside the stability region [20]. This necessarily leads to rather small time-steps, \( \Delta t \), 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, \( \nabla \cdot \mathbf{u}_c^{t+1} \), at a low enough level [22]. Furthermore, since the matrix, \( \mathbf{L} \), 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, \( \Delta t \), have to be fine enough to resolve all the relevant turbulence scales. Furthermore, the starting time for averaging, \( \tau_0 \), and the time-integration period, \( \Delta t_{avg} \), 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

\[
x_l = \frac{1}{2} \left[ 1 + \tanh \left( \frac{x_l - (x_l_{max} - x_l_{min})}{2L_{f}} \right) \right] L_{f}
\]

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 HC configuration in [24,25], for instance). In this case, the transition occurs around \( y = 0.2 \) (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 \( S = 0.45 \) (see Fig. 7, left). This value is in a rather good agreement with the experimental results obtained by Saury et al. [15] \( (S = 0.44 \pm 0.03 \text{ with } \varepsilon = 0.1 \text{ and } S = 0.57 \pm 0.03 \text{ with } \varepsilon = 0.6, \text{ where } \varepsilon \text{ is the wall emissivity}) \). The averaged Nusselt number at the cavity mid-depth is displayed in Fig. 7 (right). The profile is again rather similar to the experimental...
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=0.2 whereas in the experimental results this point is at y=0.3. Integrating the averaged local Nusselt number over the y-direction, the overall Nusselt number is determined. In this case, (Nu) = 259.2, a slightly higher value than the one obtained by D. Saury et al. [15], i.e. (Nu) = 231 ± 30 but very similar to the value obtained by means of LES, i.e. (Nu) = 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 average 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. 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, \( N = (SPr)^{0.5}/(2\pi) \), 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 [20].

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 HPC2 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 groupwork 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 (ENE2014-50577-R). X. Á. is supported by a FI predoctoral contract (FI-B-2017-00614). A. G. is supported by the Russian Science Foundation (grant 15-11-30038). 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 postdoctoral grant (JCI-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.ncki.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 in RAMS. The authors thankfully acknowledge these institutions.

References


Please cite this article as: X. Álvarez et al., HPC2: A fully-portable, algebra-based framework for heterogeneous computing. Application to CFD, Computers and Fluids (2018), https://doi.org/10.1016/j.compfluid.2018.10.034


The DNS results presented in this paper are publicly available in http://www.citeseerx.ist.psu.edu/viewdoc/doi?doi=10.1109/ICMC.2011.61