Analysis of high performance conjugate heat transfer with the OpenPALM coupler

2015 Comput. Sci. Disc. 8 015003

(http://iopscience.iop.org/1749-4699/8/1/015003)

COMPUTATIONAL SCIENCE&DISCOVERY

Analysis of high performance conjugate heat transfer with the OpenPALM coupler

Florent Duchaine[1], Stéphan Jauré[1,2], Damien Poitou[1], Eric Quémerais[3], Gabriel Staffelbach[1], Thierry Morel[1] and Laurent Gicquel[1]

[1] CERFACS, 42 Avenue Gaspard Coriolis Toulouse, France
[2] Safran Snecma, Site de Villaroche, F-77550 Moissy Cramayel, France
[3] Département Simulation Numérique des Écoulements et Aéroacoustique (DSNA), ONERA, 29 Avenue de la Division Leclerc, F-92322 Châtillon Cedex, France
E-mail: florent.duchaine@cerfacs.fr

Received 12 March 2014, revised 30 June 2014
Accepted for publication 4 May 2015
Published 27 July 2015

Computational Science & Discovery **8** (2015) 015003

doi:10.1088/1749-4699/8/1/015003

Abstract

In many communities such as climate science or industrial design, to solve complex coupled problems with high fidelity external coupling of legacy solvers puts a lot of pressure on the tool used for the coupling. The precision of such predictions not only largely depends on simulation resolutions and the use of huge meshes but also on high performance computing to reduce restitution times. In this context, the current work aims at studying the scalability of code coupling on high performance computing architectures for a conjugate heat transfer problem. The flow solver is a Large Eddy Simulation code that has been already ported on massively parallel architectures. The conduction solver is based on the same data structure and thus shares the flow solver scalability properties. Accurately coupling solvers on massively parallel architectures while maintaining their scalability is challenging. It requires exchanging and treating information based on two different computational grids that are partitioned differently on a different number of cores. Such transfers have to be thought to maintain code scalabilities while maintaining numerical accuracy. This raises communication and high performance computing issues: transferring data from a distributed interface to another distributed interface in a parallel way and on a very large number of processors is not straightforward and solutions are not clear. Performance tests have been carried out up to 12 288 cores on the CURIE supercomputer (TGCC/CEA). Results show a good behavior of the coupled model when increasing the number of cores thanks to the fully distributed exchange process implemented in the coupler. Advanced analyses are carried out to draw new paths for future developments for coupled simulations: i.e.

optimization of the data transfer protocols through asynchronous communications or coupling-aware preprocessing of the coupled models (mesh partitioning phase).

Keywords: code coupling, massively parallel, Large Eddy Simulation, conjugate heat transfer

1. Introduction

Determination of heat loads such as wall temperatures and heat fluxes is a key issue in gas turbine design [1, 2]: the interaction of hot gases and reacting flows with colder walls is a key phenomenon in all combustion chambers and is actually a main design constraint in current gas turbine developments. With the constant increase of computing power, numerical simulations of the thermal interaction between fluid flows and solids offer new design paths to diminish development costs through important reductions of the number of experimental tests. To determine mean heat loads on structures, many authors use conjugate heat transfer (CHT) where the fluid and solid equations are resolved simultaneously to predict the temperature and heat flux distributions in the system with a high level of fidelity. CHT is a difficult field and most existing tools are developed for chained (rather than coupled), steady (rather than transient) phenomena thanks to Reynolds-averaged Navier–Stokes (RANS) solvers [1, 3, 4]. The accuracy of the CHT predictions largely relies on the computational fluid dynamics (CFD) method. Recent contributions based on Large Eddy Simulation (LES) [5–7] provide promising results especially for the prediction of heat transfer in complex geometries [8–12].

Use of an unsteady LES flow solver to resolve such problems raises several complexities to address for CHT. LES requires high mesh resolutions to accurately capture the flow physics. It is also more CPU consuming than RANS methods to converge spatial and temporal statistics. These specificities imply the use of specific strategies to accelerate the convergence toward steady heat transfer problems as well as efficient methods to use existing high performance architectures to decrease the restitution times of such coupled simulations. Previous studies have proposed guidelines to reach stable convergence of CHT problems with LES based on time desynchronization of convection and conduction as well as the use of high frequency information exchanges between the different physics [8, 12, 13]. There are two basic approaches to numerically solving CHT problems. The first one is a direct coupling approach where the different physics are solved simultaneously in a large system of equations by a monolithic solver [14–18]. The second approach consists of solving each set of equations separately with dedicated solvers that exchange interface conditions through a coupler [3, 19–23]. The last solution adopted here has the advantage of using existing state-of-the-art codes to solve fluid and solid equations.

In this context and with the convergence recommendations, the resolution of CHT problems puts a lot of pressure on the tool used to couple the solvers. Several communities have investigated the use of code couplers in many different areas ranging from climate studies to industrial applications. These communities are now faced with the challenge of running the coupled applications with highly loaded codes on massively parallel machines where the solvers exchange a large amount of data at a high rate of exchange. The strategy investigated in this work to address these issues relies on recent developments made in a generic parallel

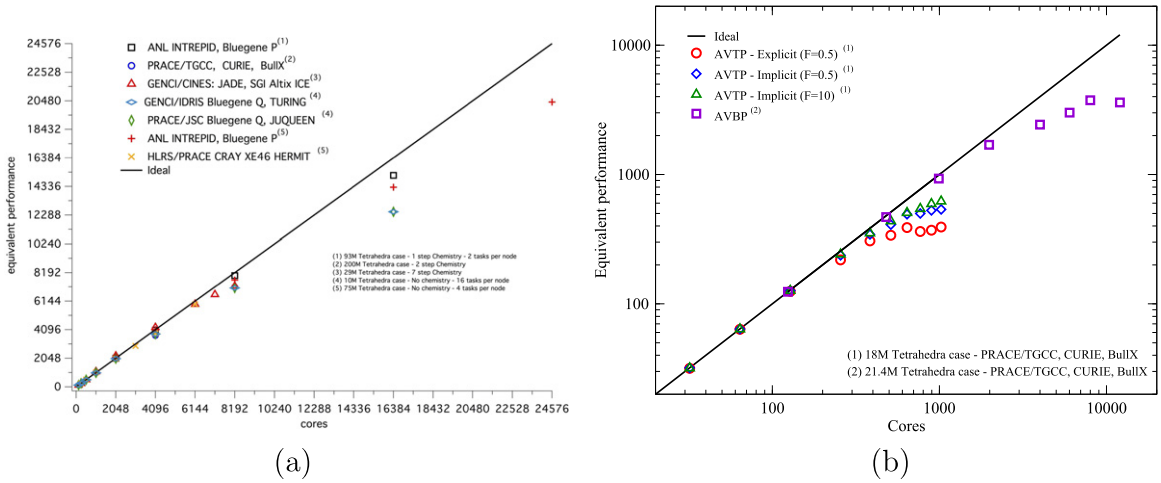

Figure 1. Strong speed-up curve obtained for (a) AVBP on different machines and physical configurations and (b) AVBP and AVTP on the CURIE/TGCC supercomputer with the target configuration of the study.

coupler [24]. The first section presents the fluid and solid solvers. Then, implementation details on the coupling library are provided. Finally performance tests carried out up to 12 288 cores on the CURIE supercomputer (TGCC/CEA) are proposed and analyzed.

2. Presentation of the flow and conduction solvers

This section presents the fluid and solid solvers used to construct the CHT tool. Both solvers have a mesh partitioning-based parallelism. After a short description of their functionalities, parallel performance of each solver is given.

2.1. The fluid solver AVBP

The AVBP project was historically motivated by the idea of building a modern CFD software with high flexibility, efficiency and modularity. Recent information about this flow solver can be found in [25]. The solver is routinely used and developed in the frame of cooperative works with a wide range of industrial companies. AVBP is capable of handling hybrid unstructured grids to ease complex geometry grid generation and adapted to high order numerical schemes in time and space. AVBP solves the compressible Navier–Stokes equations and focuses on unsteady turbulent flows (with and without chemical reactions) for internal flow configurations. The prediction of turbulent flows is based on the LES sub-grid scale closure problem [5]. The data structure of AVBP employs a cell-vertex finite-volume approximation. The numerical methods are the Lax–Wendroff scheme [26] or finite-element-type low-dissipation Taylor–Galerkin [27, 28] discretizations combined with linear-preserving artificial viscosity models. The AVBP library includes integrated parallel domain partitioning and data reordering tools, handles message passing and includes supporting routines for dynamic memory allocation, parallel input/output and iterative methods.

Typical strong speed-up obtained with AVBP is illustrated in figure 1(a). This flow solver shows excellent scalability up to 8192 cores. Results are still good for 24 000 cores (efficiency

is around 85%). The decrease in performance for a very large number of cores (point around 16 000 cores) underlines a physical limit often encountered on massively parallel applications. Indeed for very large numbers of cores the ratio between computation time and communication time is directly proportional to the problem size (number of grid points and unknowns). For this specific illustration, the configuration tested with 16 000 cores corresponds to a calculation without enough cells per core or a too low computational workload for each core compared to the amount of exchanges needed to proceed with the CFD computation. It shows that a given task is limited in terms of scalability and no increase in performance is expected beyond a given number of cores. When writing this paper, improvements of the fluid solver have been pushed forward allowing better speed up on more cores.

2.2. The solid solver AVTP

The AVTP solver has been written based on the data structure of AVBP. The solver inherits from the mesh capability and the computational performances of AVBP. AVTP solves the time dependent energy conservation equation. The heat diffusion follows Fourier's law and the solid solver takes into account local changes in heat capacity and conductivity with temperature. The second order Galerkin diffusion scheme [29] for spacial discretization comes from the AVBP solver. Time integration is done either with an explicit or an implicit first order forward Euler scheme. The resolution of the implicit system is done with a parallel matrix free conjugate gradient method [30].

AVTP resolves only one equation with only one operator: i.e. at least five times less equations and two times less operators than AVBP for a 3D configuration without combustion. The number of operations per iteration done by AVTP is thus very low compared to AVBP. A typical strong speed-up obtained with AVTP is illustrated in figure 1(b) for explicit and implicit schemes. Due to the small number of operations per iteration, the scaling of the explicit integration becomes poor at about 500 cores and then saturates. For the same Fourier number (F = 0.5 at which the explicit scheme is limited due to stability reasons), the implicit scheme shows better scaling performances due to the fact that it needs more operations per iteration than the explicit scheme. The price for one iteration on one processor is thus higher when using the implicit scheme compared to the explicit one. Increasing the Fourier number, the number of operations per iteration increases (the method needs more sub-iterations to converge) and thus the scaling appears to be better (F = 10 on figure 1). Note, however, that using larger Fourier numbers increases the time step resulting in a global decrease of consumed CPU time to simulate a given physical time, which is why implicit schemes are preferred for this type of solver. Finally, as the number of partitions increases (i.e. increasing the number of cores), the more the conjugate gradient algorithm needs sub-iterations to converge, which increases the overall number of operations per iteration and thus participates in the decrease of the overall efficiency of the solver. After 750 cores, the efficiency of the implicit solver at F = 10 is hence less than 75%.

3. Presentation of the coupler

The OpenPALM software is a code coupler, i.e. a library of functionalities that facilitate the scheduling of existing components' execution sequentially or concurrently as well as the exchange of data between these components. This is achieved in part via a collection of

primitives that are called in the codes as well as with more complex mechanisms for application scheduling. OpenPALM aims at implementing a general tool allowing one to easily integrate high performance computing applications in a flexible and evolutive way proposing a solution to the balance between performance, software reuse and numerical accuracy. OpenPALM is mainly composed of three complementary components: (1) the PALM library, (2) the CWIPI library and (3) the graphical interface PrePALM. As the application programming interface is available in Fortran and C/C++, OpenPALM can couple codes written in different languages. In the following, each component is described briefly.

3.1. The PALM library

PALM has originally been designed for oceanographic data assimilation algorithms, but its domain of application extends to every kind of scientific application [31]. In the framework of PALM, applications are split into elementary components that can exchange data though MPI communications. The main features of PALM are the dynamic launching of the coupled components, the full independence of the components from the application algorithm, the parallel data exchanges with redistribution and the separation of the physics from the algebraic manipulations performed by the PALM algebra toolbox. It can be defined as a dynamic coupler for its ability to deal with situations where the component execution scheduling and the data exchange patterns cannot be entirely defined before execution.

3.2. The CWIPI library

Based on the BFT and FVM libraries [32], CWIPI [33] aims at providing a fully parallel communication layer for mesh based coupling between several parallel codes with MPI communications. Like most existing coupling libraries for multi-executable paradigms [13, 22, 23, 34], CWIPI is a static coupler in the sense that all the components of the simulations are started at the beginning, exchange data during the run phase and finish together at the end. Coupling is made through 1D, 2D or 3D exchange zones that can be discretized in different ways in the coupled codes. The library takes into account all types of geometrical elements (polygon, polyhedral) with an unstructured description. CWIPI functionalities involve the construction of the communication graph between distributed geometric interfaces through geometrical localization, interpolation on non-coincident meshes, exchange of coupling fields for massively parallel applications as well as visualization file building. The CWIPI library is the part of OpenPALM that is mostly used in this study and is thus explained in more detail in the next section.

3.3. The graphical interface PrePALM

The graphic user interface, called PrePALM, is a portable Tcl/Tk application. The user describes entirely through this user interface the execution scheduling, the parallel sections, the data exchange patterns and the algebraic treatments [24]. It ends up providing the input file to the coupler executable as well as the source code for the wrappers of the coupled component which take care of the set-up of the communication context with no need for changes in the component sources. The same graphic tool can be used at run time to monitor the simulation status and to provide post-mortem statistics on the memory and CPU time resources used by the different components.

4. High performance mesh based coupling

Code coupling is an appealing method to develop multiphysics and multicomponent applications. However, if it is done incorrectly it can become a performance pitfall and render useless the efforts invested to optimize each individual code. There are at least two important aspects to take into account to manage efficient code coupling in an HPC context: (1) reducing the overhead of data transfer between the solvers and (2) maintaining a low global processor idle time. Indeed, and unless both codes have perfectly equal CPU per iteration times, the fastest code will have to wait for the others. Having a good load balancing is the key to maintaining a low idle time and thus reducing CPU waste.

The first point requires the most attention and a direct point-to-point communication between each solver's processors is proposed. Also because non-matching grids are used, a parallel interpolation method is required. The algorithm consists of two parts: the initialization or setup phase, i.e. where the communication routes and the interpolation coefficients are computed, and the run-time phase, or how inter-code synchronization is actually executed. The first phase is done just once per coupled simulation except if the geometries are mobile.

For the description of these phases, let us consider that solvers A and B are linked by a coupling through their respective discretized coupled interface I_A and I_B.

4.1. Inter-code communication scheme (ICCS) determination

During the initialization phase, OpenPALM creates an internal communicator for each code that replaces the MPI_COMM_WORLD communicator. Then, for each coupling defined by the user (i.e. link between two solvers), inter-communicators are created between pairs of coupled codes. From the user's point of view, the inter-application communications are completely transparent even if the codes are parallel.

The communication route's construction, ICCS determination, consists of projecting the discretized interface I_A on I_B and vice versa as a preparation for the communication phase. To maintain full scalability, coupling massively parallel applications has to remain a distributed process not only during the run-time part, but also during the initialization part. Furthermore, distributing the workload in the initialization improves the capacity of the coupled application to handle large simulations (which is a key for future applications).

Connecting the interfaces I_A and I_B means being able to perform geometrical searches from a computational domain into the other to locate the degrees of freedom of I_A in I_B and vice versa. To keep the data distribution, the geometrical searches are performed in a parallel way by avoiding data centralization and sequential treatments. This objective faces a clear difficulty: in massively parallel CFD applications or heat transfer solvers the meshes are partitioned into sub-domains each processed by a different processor. This partitioning also applies to the coupling interfaces. As the partitioning algorithm is usually not aware of the coupling process, the different distributions have no reason to match, leading to complex associations between interface processors of both solvers. To address these specific difficulties the CWIPI algorithm is composed of an optimized three-level location method (algorithm 1): the first level is the partition number of the mesh, the second one is the cell number in the selected partition, and the next one is the mean value computed in the selected cell.

As the process is fully symmetric, let us consider that code A is the source code (where the data is localized for the interpolation) and code B is the target code. The corresponding

interfaces I_A and I_B are partitioned on processes, leading to n_A sub-interfaces noted I_A^n and n_B, sub-interfaces I_B^m with $n \in [1, n_A]$ and $m \in [1, n_B]$. It is worth noting that the number of sub-interfaces n_A (respectively n_B) is lower or equal to the total number of partitions of code A (respectively B). Only the processes that contain a sub-interface are involved in the projection algorithm.

The location algorithm is parameterized to adjust the size of the bounding box around the source process sub-interface I_A^n for step #2 (algorithm 1) as well as around the source cells for step #4. This parameter takes the form of a tolerance: increasing it helps to locate points when geometries are not exactly matching. Note, however, that increasing tolerance results in an increase of the time requested by the location algorithm to converge.

Algorithm 1. Inter-code communication scheme (ICCS) determination algorithm

Step 0:
- Each partition n of the source code defines its discretized source sub-interface I_A^n to the coupler (nodes coordinates and connectivity of the cells).
- Each partition m of the target code defines its discretized target sub-interface I_B^m to the coupler (nodes coordinates and connectivity of the cells).

Step 1: Each process of the the source code defines a surrounding box of its partition I_A^n
Step 2: Each process of the source code checks for geometrical intersections of its surrounding box of sub-partition I_A^n with target nodes of the different target sub-interface I_B^m
return Determination of a reduced number of target nodes per source process n
return Construction of a first communication graph between source and target processes
Step 3: Each process of the source code classifies the previous target nodes in an octree structure to optimize the next research step
return Octree structure containing the target nodes
Step 4: Each source process defines a sub-box per mesh element of its sub-interface I_A^n
Step 5: Each source process checks the intersection between each source cell sub-box of I_A^n and the target nodes classified in the octree
return Determination of a limited number of candidate target nodes per source cell
Step 6: For each target node, the source process identifies the closest element of the source sub-interface I_A^n and defines the final communication graph
return Final communication graph from the source processes to the target ones

4.2. Communication phase

The communication phase consists of the interpolation of the fields and the exchange of the data between the solvers. The data can be stored either at the center of the cells for cell-centered solvers or at the nodes for cell-vertex solvers. Interpolation is done directly by the source solver via linear methods, i.e. barycenter interpolation with $\mathbb{P}_1$ elements (triangular in 2D and tetra in 3D), implying a spatial accuracy of 2. Note that the user can customize the interpolation with callback definition. To ensure communication scalability the communication scheme between each solver is based on direct point-to-point communications between the processors which share a common interface following the communication graph ICCS setup of the previous phase. Each processor generally has several counterparts, which have to provide a portion of their data field.

Two communication schemes exist in the library: (1) synchronous and (2) asynchronous. In the synchronous mode, each process of code A that treats a sub-interface I_A^n is involved in a

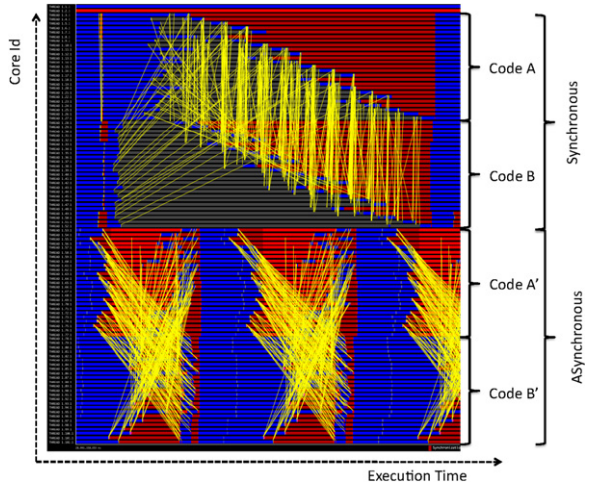

Figure 2. MPI traces of the the synchronous and asynchronous communication schemes on a coupling toy. Codes A, B, A' and B' are running on $p = 25$ cores. Blue zones correspond to computations, red to MPI exchange, gray to core waiting and yellow traces are communications.

loop of communications with processes of code B that share partially this sub-interface. The bounds of this loop are obtained thanks to the ICCS determination phase and follow the natural order of the process numbering. The exchanges are based on the MPI_Sendrecv primitive so that they can be mono- or bi-directional. This primitive is a blocking one implying that an exchange between process n of code A and process m of code B has to be finished before starting the exchange between n and $m + 1$. This method is not well optimized when a very large number of cores is involved in the coupling. Concerning the asynchronous mode, the exchanges are based on loops around the primitives MPI_Issend for sending and MPI_Irecv for reception. The completion monitoring of the exchanges is achieved thanks to loops around primitives MPI_Wait. In this mode, communications can overlap in a fully transparent way which is prone to be better performing than the synchronous mode. The other advantage of the method is its potential to overlap communication times with other treatments in the codes that do not affect the exchanged fields.

To illustrate the differences of performance between the synchronous and asynchronous communication modes, a simple toy is used. In this toy, codes A and B exchange data with the synchronous mode and codes A' and B' are coupled with the asynchronous method. Each code is running on the same number of cores (p) and performs 100 ping-pong exchanges of one quantity. All the p cores of all the solvers participate in the coupling that is performed on a 3D surface. Note that A (respectively B) and A' (respectively B') are two instances of the same source code that perform only data exchanges.

Figure 2 illustrates the communication patterns of synchronous and asynchronous communication schemes with $p = 25$. Only a zoom of the execution traces obtained with Extrae/Paraver [35] is provided here. The loss of resources due to the ordered loop of the synchronous mode is clearly evidenced by the large gray zones (representing waiting periods of the cores) for both codes, the steps forming the communication pattern. By contrast, the asynchronous method exhibits almost no waiting period for the cores. As a result, codes A' and B' perform at least two ping-pong exchanges when codes A and B perform only one for the present case.

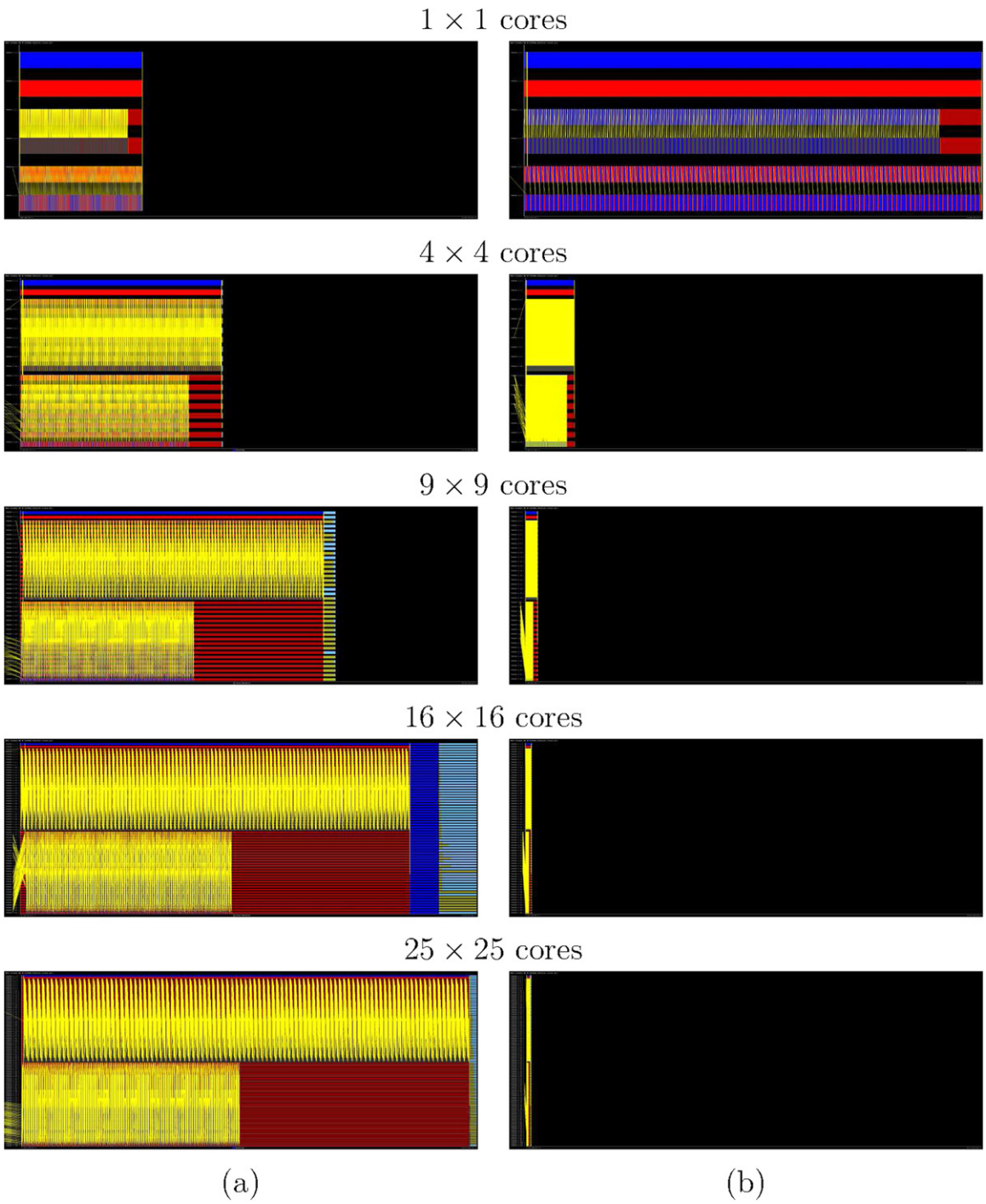

Figure 3. MPI traces of the the synchronous and asynchronous communication schemes on a coupling toy. Codes A, B, A' and B are running on $p = [1, 4, 9, 16, 25]$ cores. (a) The size of the exchange array has a fixed size per core and (b) the global size of the exchange array is the same for all the process counts. The abscissa CPU time is scaled on a 25×25 case for (a) and on a 1×1 case for (b).

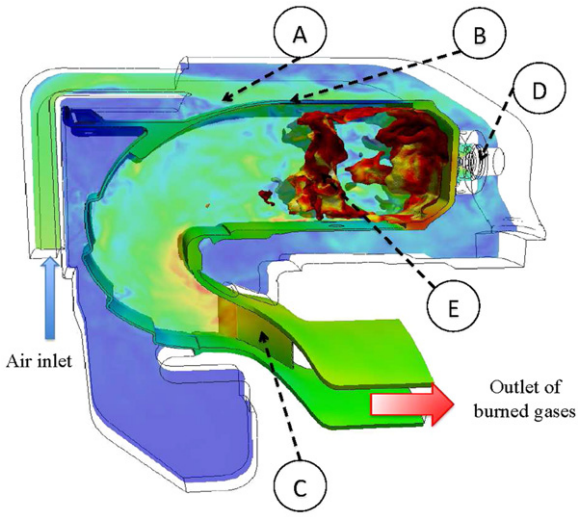

Figure 4. View of fluid and solid models of the industrial configuration. Only one sector of the annular combustion chamber is investigated.

Figure 3 presents in the same form as figure 2 the whole communication phase through Extrae/Paraver traces for two situations:

(a) increasing the number of cores per code while keeping the same amount of data to exchange per core,

(b) increasing the number of cores per code while maintaining the global size of the exchange array that is distributed on p cores.

In the first situation (a), the total amount of data exchanged between the codes increases linearly with p. The second situation (b) corresponds to a realistic computation where the meshes of the solvers are decomposed on the available cores, reducing the amount of data to exchange per computing core. Case (a) shows that increasing the number of cores per code while maintaining the size of the local array to exchange fixed leads to an important increase of the time taken by synchronous exchanges. Asynchronous communications on the other hand exhibit an almost constant time for the 100 ping-pong exchanges. When going to a more realistic situation, case (b) shows that increasing the number of cores per code while decreasing the amount of data exchange per core leads to a drastic decrease of the execution time taken to perform the 100 ping-pong exchanges. In all the core counts the performances of the asynchronous mode are better than the synchronous one.

5. Application to an industrial combustion chamber

The configuration of interest is a sector of an annular helicopter combustion chamber (figure 4), including the secondary air flow (A), the flame tube (B) and the high pressure distributor (C). The flame tube is fed with air and kerosene through an injector (D). The flame stabilizes in the primary zone (E). The thermal problem studied here by CHT consists in the determination of the temperature of combustor wall as well as of the stator.

The fluid domain is discretized with 3.8 million nodes and 21.4 million tetrahedra. To describe the flame with sufficient accuracy, the mesh is refined in the primary zone where the

flame stabilizes. The solid domain is discretized with 3.8 million nodes and 18.2 million tetrahedra. The high number of solid cells is linked to a resolution constraint that requires at least four elements in the wall thickness. The conjunction of very thin walls with the use of tetrahedra leads to a large number of elements.

Computations are done on the CURIE supercomputer, owned by GENCI and operated at the TGCC by CEA. CURIE offers different fractions of x86-64 computing resources. Thin nodes among the 5040 B510 bullx nodes of the architectures are used in the present study. Each node is composed of two eight-core Intel processors Sandy Bridge EP (E5-2680) with 2.7 GHz, 64 GB and one local SSD disk. A total of 80 640 cores are availables on the machine. The coupling has been tested on 768 processors, i.e. 12 288 cores.

The strong speed-up obtained with AVBP on the present configuration is shown on figure 1(b). The performance of the solver is very good up to 2048 cores. Afterward, a significant loss of performance is observed. This is directly linked to two things: (1) the MPI_allreduce that are not optimized in the version of Bullxmpi used for these tests and (2) the size of the fluid mesh which does not contain enough degrees of freedom to reach good scaling properties up to 8000 cores. The strong speed-up obtained with AVTP on the present configuration is also shown on figure 1(b). The curve is the one already discussed during the AVTP description. A good scalability is observed up to 650 cores when using the implicit scheme with a Fourier number $F = 10$ as in the present coupled computations.

6. Parallel efficiency of the CHT model

This section first presents the strong scaling analysis of the coupling between AVBP and AVTP from 128 to 12 288 cores. In the first part, the simulation conditions are detailed. Some important features resulting from the application that directly impact the efficiency results are underlined. The performance of the coupled application up through 12 288 cores is analyzed. Finally a second set of tests on a reduced number of cores is used to compare synchronous and asynchronous communication modes as well as a deeper performance analysis of the coupled model.

6.1. Strong scaling analysis

6.1.1. Simulation conditions. The physics and the numerics of the coupling methodology to reach a steady thermal state of the solid are detailed in [8, 13]. The main idea relies on a desynchronization of the temporal evolutions in the fluid and solid with a high frequency exchange to reduce the CPU cost while ensuring stability of CHT computations. It results that for the present case, both solvers run in parallel and exchange data at a fixed frequency corresponding to 20 times steps of the fluid code and one time step of the solid solver.

Table 1 presents the repartition of cores between AVBP and AVTP for the nine cases tested. Knowing the performances of the codes on the target machine, these repartitions aim at not slowing down the fluid code which is the more CPU greedy in case of a non-perfect synchronization at meeting points (i.e.: the cores of the solid solver wait for the fluid ones for data exchange). The performances of the coupling are compared to AVBP performances in the following of the paper. Note that the driver of OpenPALM runs on one core and that AVBP always runs on a whole processor count to avoid processor sharing between AVBP and AVTP.

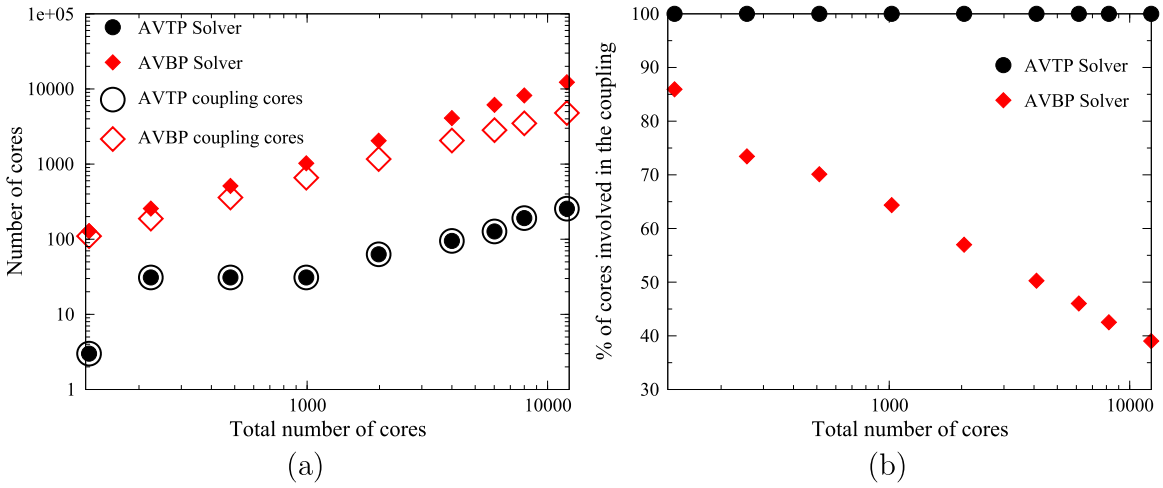

Figure 5. (a) Repartition of the total cores allocated to the solvers as well as evolution of the number of cores involved in the coupling and (b) corresponding percentage of cores involved in the coupling process for each solver.

Table 1. Repartition of cores between AVBP and AVTP for the coupled simulations. The OpenPALM driver takes one core.

Case #	1	2	3	4	5	6	7	8	9
AVBP	124	224	480	992	1984	4000	6016	8000	12 032
AVTP	3	31	31	31	63	95	127	191	255
OpenPALM	1	1	1	1	1	1	1	1	1
Total	128	256	512	1024	2048	4096	6144	8192	12 288

AVBP treats around 0.48 million cells on its discretized coupling interface and AVTP treats about 1.7 million elements. The core distribution among the solvers as well as the number of cores on which these coupling interfaces are partitioned for the 9 cases are shown in figure 5. It is worth noting that 100% of the solid cores are involved in the coupling process for all cases. On the other hand, the proportion of AVBP processes involved in the coupling interface decreases almost exponentially when the number of cores increases. This behavior is linked to the ratio between the volume of the configuration and its surface which is higher in the case of the fluid solver. It is important to note that mesh partitioning is managed independently in each solver resulting in non-conformal patterns of the partitioned discretized surfaces.

It is of interest to analyze the evolution of the distribution of the number of mesh elements in the sub-interfaces as the number of cores increases. Indeed, a reduced amount of AVBP cores have more tasks to perform than the others because of the exchanges with AVTP. The way this additional load is distributed and its evolution when the number of processing cores increases, is particularly important to ensure scaling of the coupled application. To analyze these distributions, figures 6 and 7 show the histograms of the number of cores treating n_c cells (these histograms are constructed based only on the cores with coupling cells). Abscissa scales are logarithmic to facilitate the interpretation. An ideal configuration would feature homogeneous distributions with all cores having the same amount of coupling cells leading to the same amount of work or communications to perform. Concerning the fluid solver AVBP,

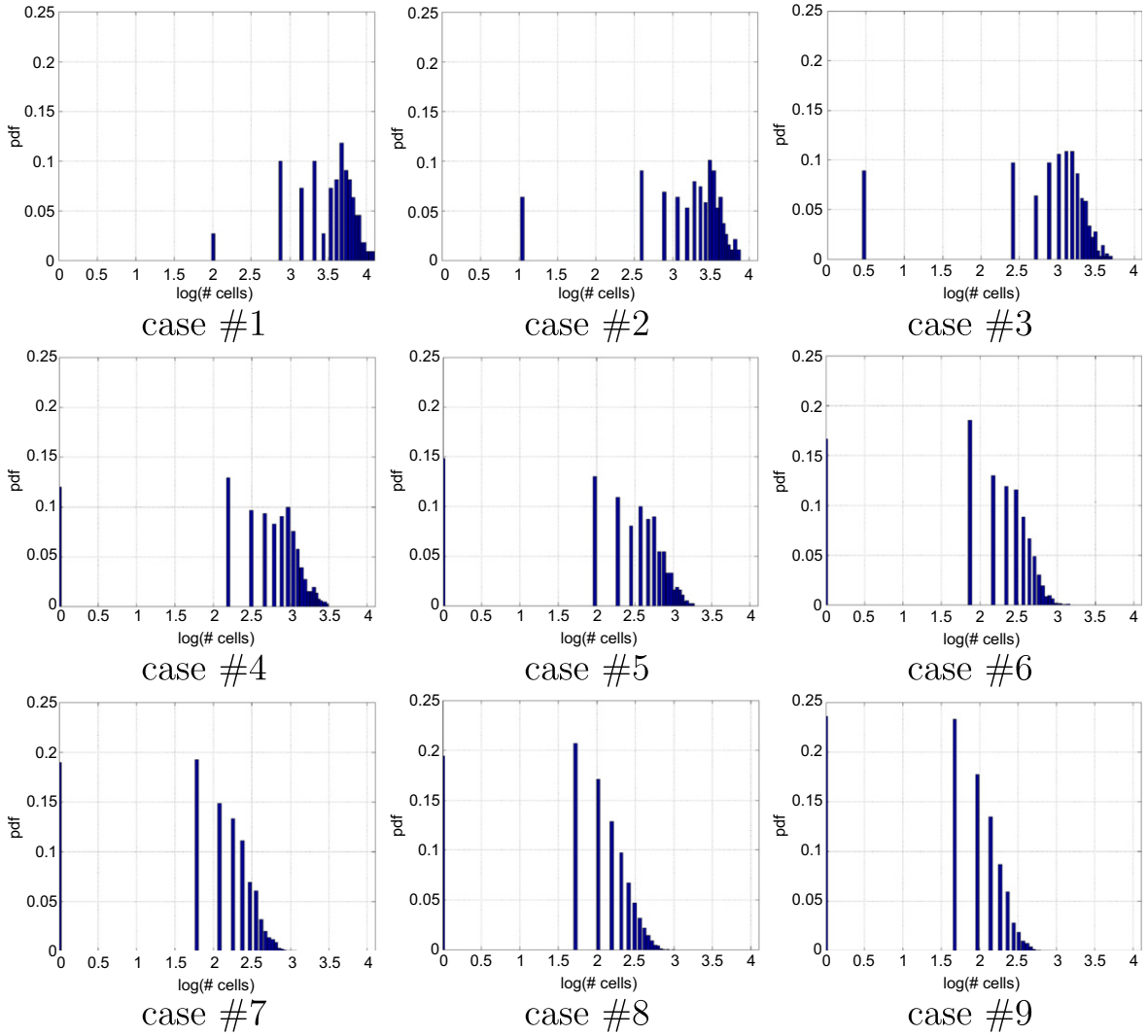

Figure 6. Histograms of the distribution of coupling cells per AVBP core depending on the total number of cores of the coupled simulations.

the histograms at low number of cores are rather spread out with distinct peaks, reflecting an inhomogeneous partitioning of the coupled discretized interface among the computing cores. Increasing the number of cores narrows the range of the histograms around the small number of cells. A significant peak emerges indicating a homogenization of the distribution of coupling cells on the cores. Nevertheless, the peak at very low number of cells remains highlighting the participation of cores in the process of inter-code exchanges for a small number of information. From cases #3 to #9, the peak has the same order of magnitude as the main peak of the distribution.

As far as AVTP is concerned, case #1 exhibits a rather homogenous distribution on the three cores. Then all the distributions show a multimodal profile with a marked peak in the range of small numbers of coupling cells per core.

The different analyses in this section underline phenomenological issues resulting from the coupler as well as the CHT application setup. Some of these results consist of potential

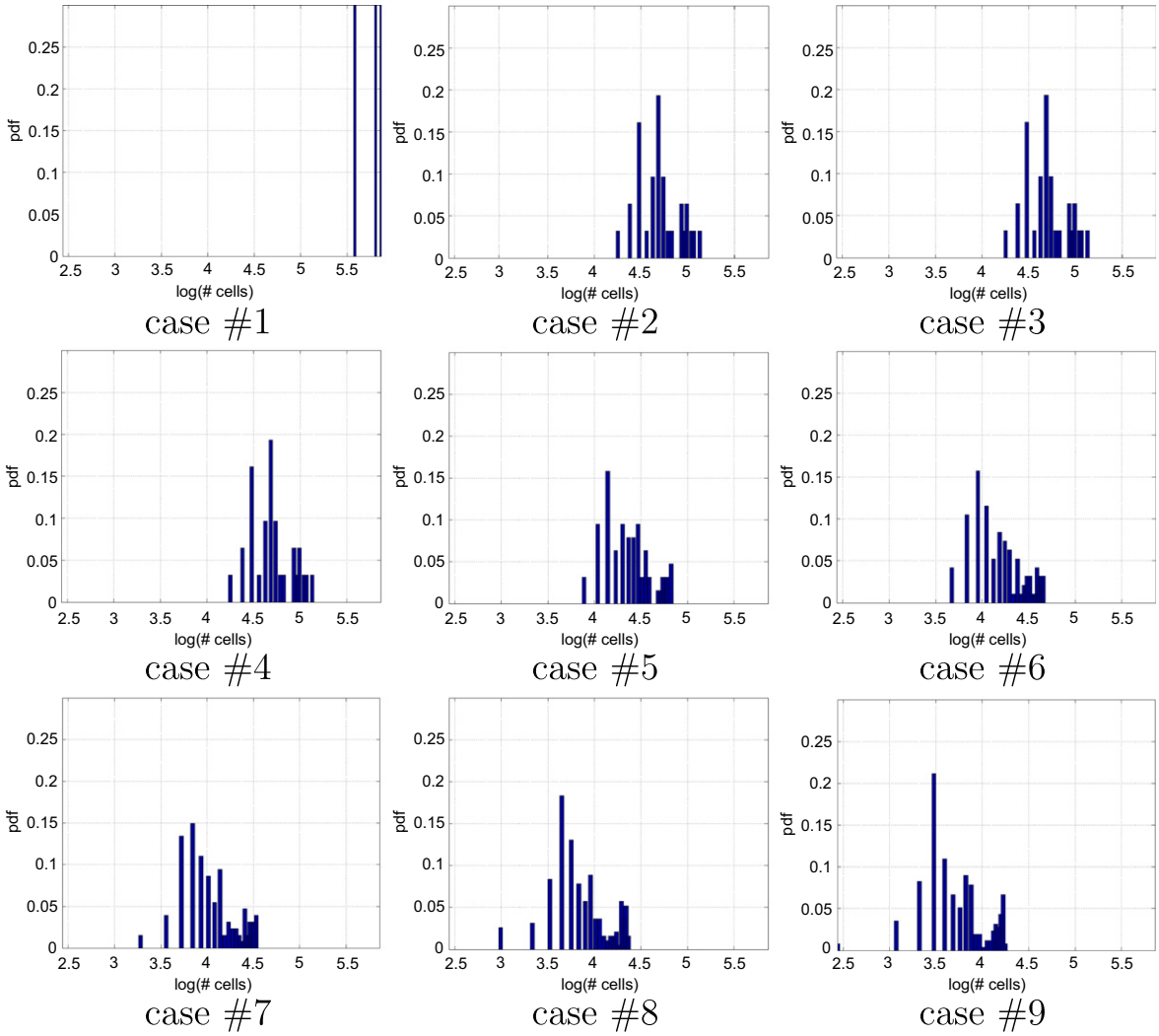

Figure 7. Histograms of the distribution of coupling cells per AVTP core depending on the total number of cores of the coupled simulations.

weaknesses for performance results: imbalance between the number of cores allocated to the solvers, all the cores of the AVBP solver not participating in the coupling creating a difference of computing load inside the code itself; imbalance repartition of the sub-interface among the coupling cores on each solver; complex communication scheme with a lot of connections between the processes of the two solvers, and finally a combination of these features. Having identified these behaviors, the next section analyzes the strong scalability of the application.

6.1.2. Performances of the coupled application with synchronous communications. The time needed to construct the ICCS is presented in figure 8(a). Except for case #1, this time is almost constant when the number of cores increases. For case #1, the ratio of cores involved in the coupling of AVBP over the one involved in the coupling of AVTP is the most important. Indeed, this ratio translates partly the imbalance of cores that have to communicate together leading to a very poor efficiency of the algorithm. Increasing the number of cores for AVTP from case #1 to case #2 leads to a drastic reduction of the consumed time. From cases #2 to #4,

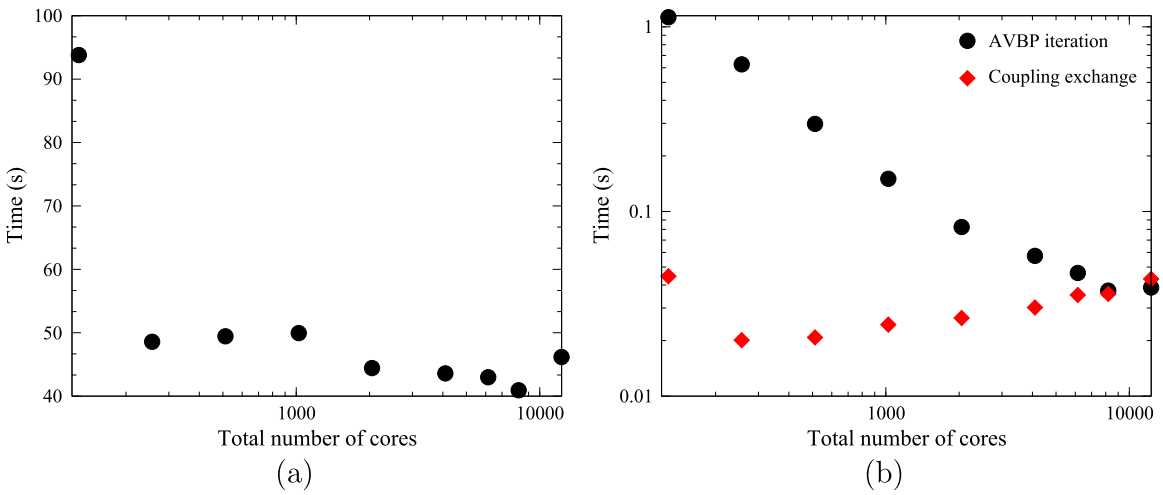

Figure 8. (a) Time taken by the inter-code communication scheme determination algorithm. (b) Comparison of the time for one iteration of AVBP and the time to do one exchange as a function of the total number of cores of the coupling.

the number of cores allocated to the AVTP coupling interface is fixed while the number of AVBP cores increases. It results in a slight increase of the time spent to construct the communication scheme. Then, the ratio of coupling cores between the solvers is almost constant resulting in a slight decrease of the time consumed by the ICCS algorithm up to 8000 cores. From these analyzes, one can note that the scaling of the ICCS algorithm largely depends on the number of processes involved in the coupling on each side of the coupled model. Increasing the number of cores of just one code is not sufficient to ensure a good scaling of this coupling phase.

Figure 8(b) presents the evolution of the time taken by an AVBP iteration compared to the time of an exchange between the solvers as a function of the total number of cores of the coupled system. The exchanges are made in the synchronous mode. As previously mentioned, the fluid solver exhibits a rather good scaling until 4000 cores. Above 8000 cores, the time of an iteration reaches a plateau. The time needed for the data exchange is almost constant whatever the number of cores. As for the first phase, the time of the communication step exhibits an important decrease from case #1 to #2. This reveals that the balance of processes that are coupled continues to play an important part in this phase. Then, the communication time slightly increases as the number of cores grows. Until several thousand of cores, the time requested by an exchange is several order of magnitude lower than the time of an AVBP iteration. Then both times are of the same order. As the coupling is done every 20 iterations of the fluid solver AVBP, the global scaling of the application presented on figure 1(b) is conserved. The communication phase is thus fully transparent in term of restitution time for the user.

6.2. Additional performance measurements

To gauge the relative performances of synchronous and asynchronous communication modes on the real geometry, the 16 cases presented in table 2 have been performed. This set of computational experiments aims at drawing a map of exchange times by fixing a different number of AVTP cores and increasing the number of AVBP cores. This map is shown on

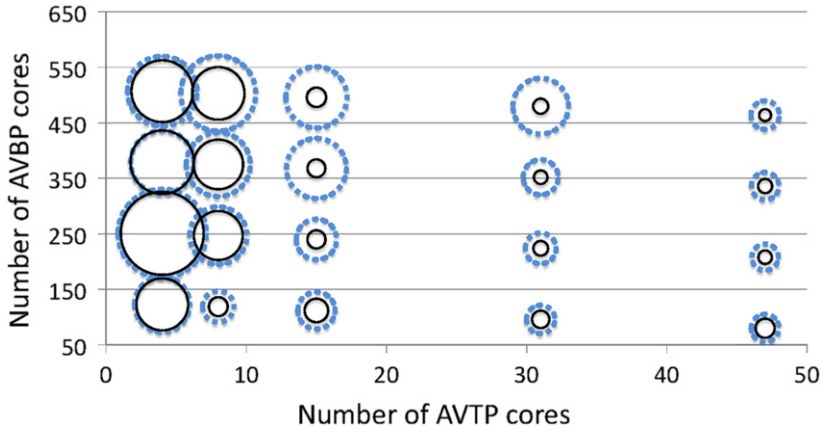

Figure 9. Comparison of synchronous (dotted lines) and asynchronous (continuous lines) communications. The diameters of the circles are proportional to the time spent in the communication process.

Table 2. Repartition of cores between AVBP and AVTP and time taken by one exchange for the comparisons of synchronous (sync.) and asynchronous (async.) communications modes.

# AVTP cores	# AVBP cores	Async. exch. time (s)	Sync. exch. time (s)
4	123	0119	0135
4	251	0303	0344
4	379	0177	0176
4	507	0167	0214
8	119	0016	0043
8	247	0105	0145
8	375	0109	0181
8	503	0121	0248
15	112	0025	0061
15	240	0015	0072
15	368	0014	0159
15	496	0017	0165
31	96	0013	0036
31	224	0010	0042
31	352	0008	0054
31	480	0010	0136
47	80	0016	0034
47	208	0008	0031
47	336	0009	0034
47	464	0006	0036

figure 9, comparing the times for one exchange between the solvers for the two communication modes. The first noticeable element is that asynchronous communications always outperformed synchronous ones in this example. Then, for almost all the cases, for a fixed number of AVTP cores, increasing the number of cores of AVBP leads to an increase of the exchange time between the solvers as evidenced in the previous section. This increase is much more evident

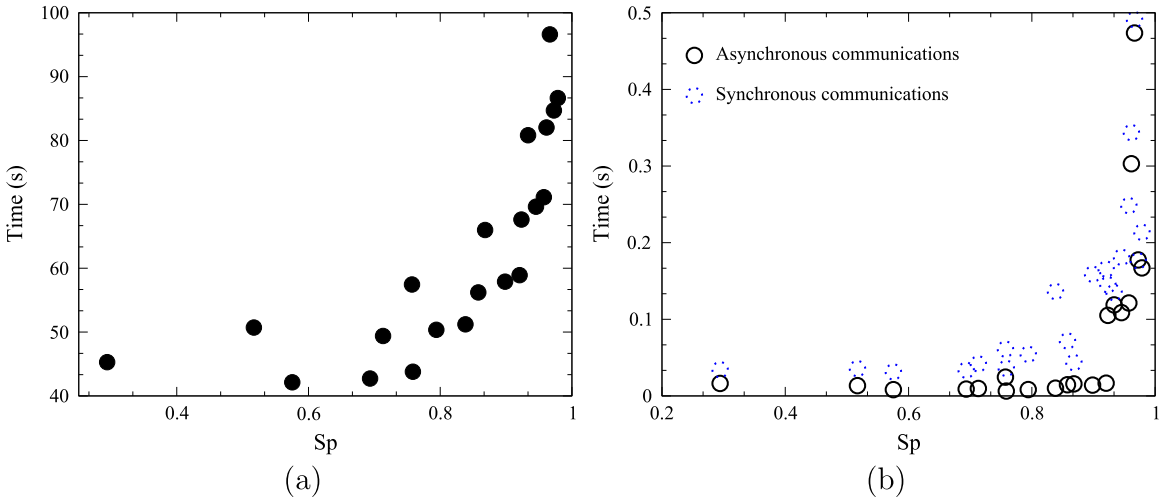

Figure 10. (a) Time taken by the inter-code communication scheme determination algorithm as a function of the sparsity parameter Sp and (b) exchange times of the synchronous and asynchronous communications as a function of the sparsity parameter Sp.

for the synchronous cases. When increasing the number of AVTP cores at almost fixed value of AVBP cores, the trend of the exchange times is to decrease. This decrease is more important for the asynchronous cases. A very important piece of information is that scalability of the exchange times depends on the number of cores of each code: increasing the number of cores of only one solver leads to a decreasing exchange time up to a given lower limit. To continue decreasing the time for exchanges, the number of cores affected by the second solver has to be increased to spread the communication loads. An important parameter that seems to drive the communication time between the solvers is the sparsity:

$$\mathrm{Sp} = \frac{\mathrm{np}_{\mathrm{AVBP}} - \mathrm{np}_{\mathrm{AVTP}}}{\mathrm{np}_{\mathrm{AVBP}} + \mathrm{np}_{\mathrm{AVTP}}} \tag{1}$$

were $\mathrm{np}_{\mathrm{AVBP}}$ and $\mathrm{np}_{\mathrm{AVTP}}$ are the number of AVBP (resp. AVTP) cores involved in the coupling interface of the AVBP (resp. AVTP) solver. Figure 10 shows that the times needed to construct the ICCS as well as to perform exchanges correlate very well with Sp. This correlation has to be explored in more details.

7. Conclusion

The different choices made for the resolution of CHT problems on complex geometries with a high fidelity solver put a lot of pressure on the tool used for the coupling. As in several communities, developers are now faced with the challenge of running coupled applications with highly loaded codes on massively parallel machines and with solvers exchanging a lot of data at a high frequency. The strategy investigated in this work to address these issues relies on recent developments made in a generic parallel coupler. The characteristics of the coupled tool including the fluid and solid solvers as well as the coupler are detailed. The tool is then applied to an industrial combustion chamber. Performance tests are carried out until 12 288 cores on the CURIE supercomputer (TGCC/CEA) and relevant parameters that influence the coupling

scalability are detailed. The coupler exhibits a very good behavior up to 12 288 cores implying that the use of HPC can drastically reduce the restitution time of coupled applications for industrial design with high fidelity solvers. Analyses of the scaling response underline the impact of imbalanced repartitions of cores among the codes, imbalance repartitions of the sub-interface among the coupling cores on each solver, as well as the complex communication scheme which can infer a lot of connections between processes of the solvers on coupling overhead. These points are independent from the coupler and can be addressed by incorporating the knowledge of the coupling in the preprocessing step of the solvers (constraint and co-partitioning). Tests on asynchronous communications show an important improvement of the scalability of the coupler indicating development paths for the future. Finally, optimizations can be envisaged to increase the performances of synchronous communications by enhancing the ordering of the communications.

Acknowledgments

The work was supported by the Fondation de Recherche en Aéronautique et Espace in the project STRASS. The TGCC is gratefully acknowledged for providing access of the CURIE machine during the *Grand Challenge*. Turbomeca is acknowledged for supporting code developments and permission to publishing results.

References

[1] Schiele R and Wittig S 2000 Gas turbine heat transfer: past and future challenges *J. Propulsion Power* **16** 583–9

[2] Bunker R S 2006 Gas turbine heat transfer: 10 remaning hot gas path challanges *Proc. GT2006. ASME Turbo Expo 2006*

[3] Garg V 2002 Heat transfer research on gas turbine airfoils at NASA GRC *Int. J. Heat Fluid Flow* **23** 109–36

[4] Mercier E, Tesse L and Savary N 2006 3d full predictive thermal chain for gas turbine *25th Int. Congress of the Aeronautical Sciences*

[5] Sagaut P 2000 *Large Eddy Simulation for Incompressible Flows (Scientific Computation Series)* (Berlin: Springer)

[6] Sagaut P and Deck S 2009 Large-Eddy simulation for aeronadymics: status and perspectives *Phil. Trans. R. Soc.* **367** 2849–60

[7] Léonard T, Gicquel L M, Gourdain N and Duchaine F 2010 Steady/unsteady Reynolds averaged Navier–Stokes and Large Eddy Simulations of a turbine blade at high subsonic outlet mach number *J. Turbomach.* **137** 041001

[8] Duchaine F, Corpron A, Pons L, Moureau V, Nicoud F and Poinsot T 2009 Development and assessment of a coupled strategy for conjugate heat transfer with large Eddy simulation: application to a cooled turbine blade *Int. J. Heat Fluid Flow* **30** 1129–41

[9] Bhaskaran R and Lele S K 2010 Large Eddy simulation of free-stream turbulence effects on heat transfer to a high-pressure turbine cascade *J. Turbul.* **11** N6

[10] Maheu N, Moureau V, Domingo P, Duchaine F and Balarac G 2012 Large-Eddy simulations of flow and heat transfer around a low-mach number turbine blade *Proc. Summer Program, Center for Turbulence Research* (NASA Ames/Stanford University)

[11] Collado Morata E, Gourdain N, Duchaine F and Gicquel L 2012 Effects of free-stream turbulence on high pressure turbine blade heat transfer predicted by structured and unstructured les *Int. J. Heat Mass Transfer* **55** 5754–68

[12] Duchaine F, Maheu N, Moureau V, Balarac G and Moreau S 2013 Large Eddy simulation and conjugate heat transfer around a low-mach turbine blade *J. Turbomach.* **136** 051015-1
[13] Jauré S, Duchaine F, Staffelbach G and Gicquel L 2013 Massively parallel conjugate heat transfer solver based on large Eddy simulation and application to an aeronautical combustion chamber *Comput. Sci. Discovery* **6** 015008-1
[14] Kao K H and Liou M S 1997 Application of chimera/unstructured hybrid grids for conjugate heat transfer *AIAA J.* **35** 1472–8
[15] Han Z X, Dennis B and Dulikravich G 2001 Simultaneous prediction of external flow-field and temperature in internally cooled 3d turbine blode material *Int. J. Turbo Jet-Engines* **18** 47–58
[16] Rahman F, Visser J A and Morris R M 2005 Capturing sudden increase in heat transfer on the suction side of a turbine blade using a Navier–Stokes solver *J. Turbomach.* **127** 552–6
[17] Ganesan V 2007 Non-reacting and reacting flow analysis in an aero-engine gas turbine combustor using cfd *SAE 2007 World Congress* no. 2007-01-0916
[18] Craig A, Vertenstein M and Jacob R 2012 A new flexible coupler for earth system modeling developed for ccsm4 and cesm1 *Int. J. High Perform. Comput. Appl.* **26** 31–42
[19] Sondak D L and Dorney D J 2000 Simulation of coupled unsteady flow and heat conduction in turbine stage *J. Propulsion Power* **16** 1141–8
[20] Papanicolaou E, Giebert D, Koch R and Schultz A 2001 A conservation-based discretization approach for conjugate heat transfer calculations in hot-gas ducting turbomachinery components *Int. J. Heat Mass Transfer* **44** 3413–29
[21] Bohn D, Ren J and Kusterer K 2005 Systematic investigation on conjugate heat transfer rates of film cooling configurations *Int J. Rotating Mach.* **2005** 211–20
[22] DeCecchis D, Drummond L and Castillo J 2011 Design of a distributed coupling toolkit for high performance computing environment *Math. Comput. Modelling* **57** 2267
[23] Valcke S *et al* 2012 Coupling technologies for earth system modelling *Geosci. Model Dev. Discuss.* **5** 1987–2006
[24] Piacentini A, Morel T, Thevenin A and Duchaine F 2011 Open-palm: an open source dynamic parallel coupler *4th Int. Conf. on Computational Methods for Coupled Problems in Science and Engineering*
[25] Gicquel L, Gourdain N, Boussuge J-F, Deniau H, Staffelbach G, Wolf P and Poinsot T 2011 High performance parallel computing of flows in complex geometries *C. R. Méc.* **339** 104–24
[26] Lax P D and Wendroff B 1960 Systems of conservation laws *Commun. Pure Appl. Math.* **13** 217–37
[27] Donea J 1984 Taylor–Galerkin method for convective transport problems *Int. J. Numer. Methods Fluids* **20** 101–19
[28] Colin O and Rudgyard M 2000 Development of high-order Taylor–Galerkin schemes for unsteady calculations *J. Comput. Phys.* **162** 338–71
[29] Donea J and Huerta A 2003 *Finite Element Methods for Flow Problems* (New York: Wiley)
[30] Frayssé V, Giraud L, Gratton S and Langou J 2005 A set of GMRES routines for real and complex arithmetics on high performance computers *ACM Trans. Math. Softw.* **31** 228–38
[31] Buis S, Piacentini A, Déclat D and The PALM Group 2006 PALM: a computational framework for assembling high-performance computing applications *Concurrency Comput.: Pract. Exp.* **18** 231–45
[32] Fournier Y, Bonelle J, Moulinec C, Shang Z, Sunderland A and Uribe J 2011 Optimizing Code_Saturne computations on petascale systems *Comput. Fluids* **45** 103–8
[33] Refloch A *et al* 2011 Cfd platforms and coupling—cedre software *Onera J. Aerosp. Lab*
[34] Joppich W and Kürschner M 2006 Mpcci—a tool for the simulation of coupled applications *Concurrency Comput.: Pract. Exp.* **18** 183–92
[35] Labarta J 2011 Trace-based tools *Performance Tuning of Scientific Applications* ed R F L D H Bailey and S W Williams (Boca Raton, FL: CRC Press) pp 87–122