Partitioned High Performance Code Coupling Applied to CFD

Florent Duchaine, Sandrine Berger, Gabriel Staffelbach, and Laurent Gicquel

Cerfacs - 42 avenue Gaspard Coriolis - 31 057 Toulouse - France
florent.duchaine@cerfacs.fr

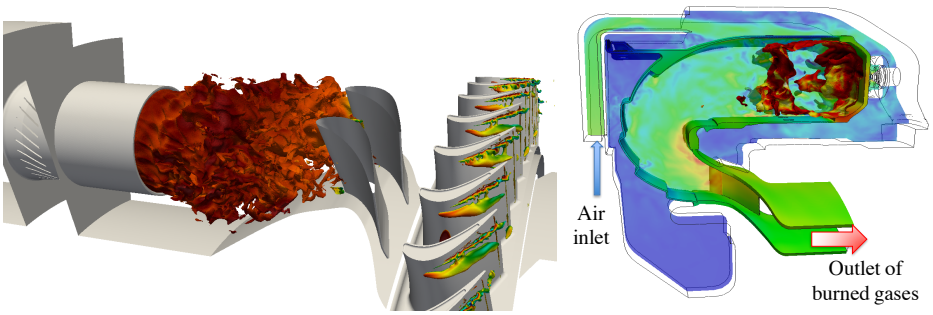
Abstract. Based on in situ observations obtained in the context of multiphysics and multicomponent simulations of the Computational Fluid Dynamics community, parallel performances of code coupling is first discussed. Overloads due to coupling steps are then analyzed with a simple toy model. Many parameters can impact the communication times, such as the number of cores, the communication mode (synchronous or asynchronous), the global size of the exchanged fields or the amount of data per core. Results show that the respective partionning of the coupled codes as well as core distributions on the machine have an important role in exchange times and thus on the total CPU hours needed by an application. For the synchronous communications presented in this paper, two main outcomes independent from the coupler can be addressed by incorporating the knowledge of the coupling in the preprocessing step of the solvers with constraint and co-partitioning as well as process placement. Such conclusions can be directly extended to other field of applications such as climat science where coupling between ocean and atmosphere is of primary importance.

1 Introduction

Today, the design of gas turbines requires to consider strong interactions between different physics as well as the components of the engine. As a result, integrated simulations involving multiphysics and multicomponents are performed both at the research level as well as in industries. With the constant increase of computing power, numerical simulations of the interactions between the compressor, combustion chamber and turbine, as well as 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. In these fields, the main idea is to jointly simulate the different parts of the coupled problems with a high level of fidelity limiting hypotheses on the boundary conditions:

- for the interactions between turbomachinery parts and combustor, inlet and outlet models of the component interfaces can be avoided by resolving the full system at once (Fig. 1-a),
- 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 (Fig. 1-b).

Recent works have shown the ability of eddy resolving methods such as Large Eddy Simulation (LES) to provide reliable results in the contexts of combustors and turbomachinery [6,14,5,10]. Using an unsteady LES flow solver to resolve such problems raises several complexities to address in the context of coupled problems. Indeed, LES requires high mesh resolutions to accurately capture the flow physics and is more CPU consuming than averaged methods to converge spatial and temporal statistics. These specificities imply to use high performance architectures to decrease the restitution times of the simulations.



(a) (b)

Fig. 1. Example of an integrated combustor/turbine simulation [3] (a), view of fluid and solid models of an industrial combustor Conjugate Heat Transfer simulation [4] (b).

There are two basic approaches to numerically solve coupled problems such as CHT. 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. The second approach consists in solving each set of equations separately with dedicated solvers that exchange interface conditions through a coupler. The last solution adopted here has the advantage of using existing state-of-the-art codes to solve fluid and solid equations. Nevertheless, it stresses the tool used to couple the solvers in terms of parallel computing performances. Several communities have investigated the use of code coupler in many different areas ranging from climate studies to industrial applications. These communities are now faced to 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 frequency.

This paper presents a feed-back on the use of coupling libraries on massively parallel systems for multiphysics and multicomponent simulations with a LES solver [7]. Based on observations monitored on real applications running on HPC systems, a toy model is constructed to identify paths of improvements on a simple controlled code.

2 In situ observations

The OpenPALM software is used in this study [4]. It 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 to easily integrate high performance computing applications in a flexible and evolutive way proposing a solution to the balance among performance, software reuse, and numerical accuracy. OpenPALM is mainly composed of three complementary components, (1) the PALM[1] library [2,11], (2) the CWIPI[2] library [13] and (3) the graphical interface PrePALM [2,11].

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 a HPC context (Fig. 2): (1) reducing the overhead of data transfer between the solvers and (2) maintaining a global processor idle time low, unless both codes have perfectly equal CPU per iteration times, the fastest code will have to wait the others. Having a good load balancing is the key to maintain a low idle time and thus reduce CPU waste. The first point requires the most attention and a direct point to point communication between each solver's processors is proposed [8]. Also non matching grids being 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. Figure 3 presents the time requested for the initialization and the run-time phases for a turbomachinery application [9] performed on Titan[3] until 132,000 cores. Globally, a decrease of both times is observed as the number of cores involved in the coupling increases. Interestingly, there are two order of magnitude difference between the two phases, the initialization being the more time consuming. These times are affected by the location algorithms, machine performance and characteristics as well as by the way external communications between solvers are handled (communication algorithm, interface partitioning).

Focusing on the communication time (run-time phase), Fig. 4 shows the exchange time as a function of the ratio between the number of cores allocated to the fluid and those allocated to the solid (in abscissa) as well as the total number of cores involved in the exchange (which increases with the bubbles size) in the

[1] Projet d'Assimilation par Logiciel Multiméthodes
[2] Coupling With Interpolation Parallel Interface
[3] Titan: Oak Ridge National Laboratory. No. 1 system of Top500 in November 2012

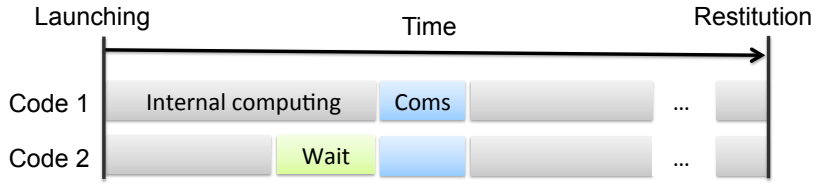


Fig. 2. Time line corresponding to a coupled simulation including two codes.

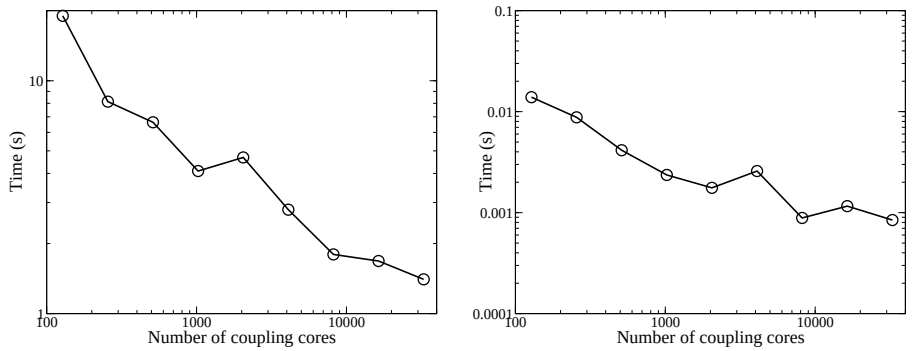


Fig. 3. Time requested for the initialization (a) and the run-time (b) phases as a function of the number of cores involved in the coupling process for a turbomachinery application [9] performed on Titan.

case of a CHT computation on Curie[4] [4]. The total number of exchanging cores (indicated by the bubble size) does not have a leading role in the variation of the communication times. Instead Fig. 4 highlights that the more the ratio of cores increases, the more communications are expensive. This points out that important unbalance in the core distribution between the solvers which may be requested to synchronize to avoid waiting as illustrated on Fig. 2 which can be detrimental for exchange time optimization. Interestingly, two points (colored in red in Fig. 4) exhibit very close core ratios with very different communications times. Neither this switching of ratio nor the corresponding total number of cores can explain by themselves the differences in the communication time between the two cases. Other underlying parameters are involved and next the section intends to give elements in this direction with a controlled toy coupled application.

[4] Curie supercomputer, owned by GENCI and operated at the TGCC by CEA.

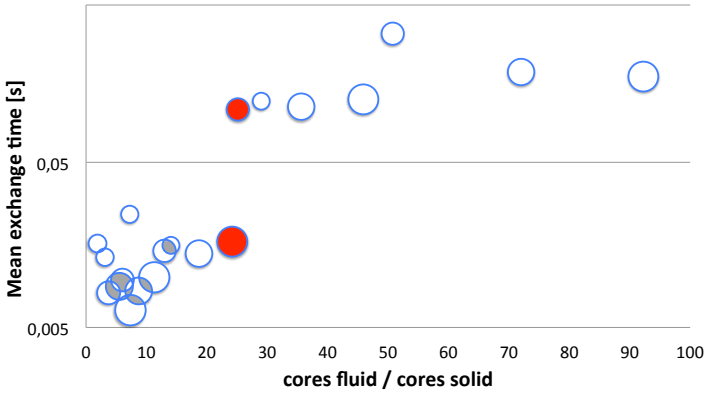


Fig. 4. Evolution of the exchange time as a function of the ratio between the number of cores allocated to the fluid and the number of cores allocated to the solid (abscissa) and the total number of cores involved in the exchange (increasing with the bubbles size). Data extracted from [4].

3 Toy model

The toy model is composed of two identical codes. In the following, quantities referring to the first and second executables are respectively indexed with the subscripts 1 and 2. Each of these entities build a square including $npts_i$ (with i the index of the solver) points distributed on N_i cores where N_i is such that $N_i = m^2$, $m \in \mathcal{N}$. As detailed on Fig. 5, the partitioning is homogeneous, i.e. each square edge is cut in the same way (which justifies the need for a number of cores such that $N_i = m^2$). The codes perform 100 data exchange ping-pongs with the OpenPALM coupler to provide statistically converged exchange times. Both the initialization and the communication phases are recorded separately. Since the initialization time mainly relies on localisation methods, the investigation focus of this study is on communication times.

The results come from computations performed on a Cerfacs-based BULL B510 Supercomputer. Each computational node includes two processors, itself composed of eight cores. The Infiniband interconnection network offers a theoretical $5GB.s^{-1}$ bandwidth between nodes. The MPI latency is lower than 1 micro-second. For the present tests, MPI communications are performed thanks to the IntelMPI library.

The influence of various parameters has been considered. This paper reports cases for synchronous communications first with the same number of cores for each executable and then with a different number of cores. The dependency of exchange time to the global amount of data on the models as well as per core is investigated by changing the number of points on the grid $npts_i$. The number of cores is denoted N_i, and the total amount of data sent by a code (in bytes, B) is denoted $datatot_i$ and the quantity of data per core is given by $dataproc_i$. No

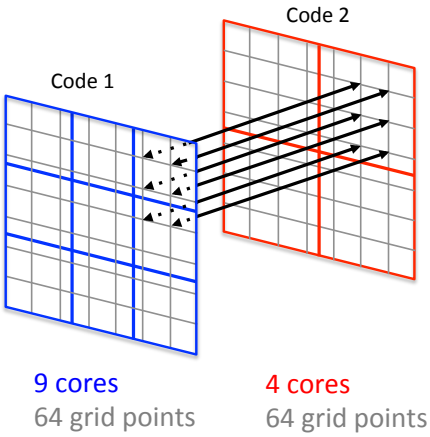


Fig. 5. Schematic of the inter-code communication toy.

placement effort is made and the MPI ranks are distributed among the available cores in a linear way, i.e. the first application is assigned to the first N_1 cores and the second one to the following N_2 cores.

The influence of the total amount of data on the grid is investigated by increasing the number of nodes that composed the grids. Figure 6-a shows the evolution of the exchange time as a function of the total amount of data on the grid for different values of core numbers $N_1 = N_2$. The curves display the same behavior in logarithmic scale with as expected the exchange time greatly increasing with the number of grid points. On the contrary, for a given number of grid points, the increase of the number of cores on which data are distributed tends to decrease the communication time. Such behavior can be mathematically modeled based on architecture parameters [1]. To explore the effect of the quantity of data per core on communications, Fig. 6-b shows the communication times arranged here as a function of the data quantity per core. This different representation of the same data highlights three groups of curves:

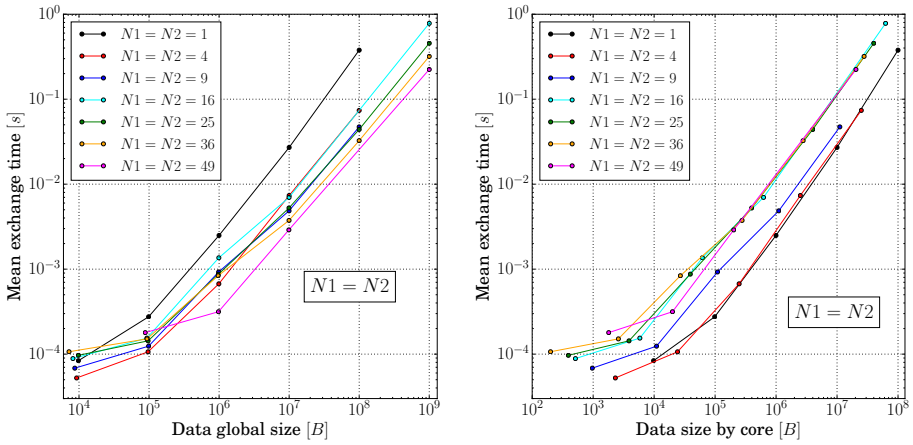
- $N_1 = N_2 = 1$ and 4
- $N_1 = N_2 = 9$
- $N_1 = N_2 = 16, 25, 36$ and 49.

These gatherings may be explained by the bandwidth variation between the various levels of the supercomputer network. The bandwidth between two computing cores of a given machine depends on their relative positioning on the network as well as on the size of the exchanged message. Three cases can be distinguished that depend on the computer communication networks used by the toy model:

- $N_1 = N_2 = 1$ and 4: the cores are distributed on the two processors of the same node. Communications are thus achieved within the same node but

on potentially different processors. They are thus relatively fast but very dependent on the exchanged message size.
- $N_1 = N_2 = 9$: the cores are mainly placed on the same node, only three cores are on a different node due to the use of one process for the coupler's driver. Even though most of the communications are intra-processor or intra-node, some exchanges are made between cores from different nodes.
- $N_1 = N_2 = 16, 25, 36$ and 49, the cores are distributed on several nodes (3 to 7 nodes depending on the case). A large part of the communications (if not all) is made between nodes. Most of the communications are thus made between cores that are quite far from each other on the network resulting in slower exchanges.

These analyses bring to the conclusion that the minimization of exchange times between coupled components can be performed by process placement on the parallel architecture. Such placement algorithm must take into account internal exchanges in the parallel models to minimize the impact on the standalone model performances.



(a) (b)

Fig. 6. Evolution of the exchange time as a function of the total amount of data on the grids $datatot_i$, for several values of the number of cores (a), Evolution of the exchange time as a function of the data size per core $dataproc_i$, for several values of the number of cores (b).

In a real coupled application, the exchange interface between two codes is rarely partitioned in the same way and/or distributed over the same number of computing cores. To investigate this point, the toy is run for cases where the number of allocated cores is different for each executable ($N_1 \neq N_2$). These tests

are performed for every possible N_1 and N_2 value combinations. The global tendencies remain the same for all cases. Therefore, for brevity, Fig. 7 presents only the results for the cases where $N_1 = 16$ and $N_2 = 16; 25; 36$. For each of them, the relative positions of the partitioning are indicated on the top of the figure. Exchange times evolve within the same range as those presented for the case $N_1 = N_2$ increasing as the total amount of data increases. However, it is worth noting that for every tested case, given a fixed number of cores N_1, every values of N_2 different from N_1 leads to communication times superior or similar to the $N_1 = N_2$ case. Cases with partitionings of the two executable grids that are either identical or quite coincident minimize the number of communications between the two codes leading to lower communication times. A smart partitioning of both domains with respect to each other could lead to lower exchange times and hence better performance of the coupled simulations. According to these observations, future work should focus on the development of co-partitioning techniques able to decrease greatly the communications time between solvers [12].

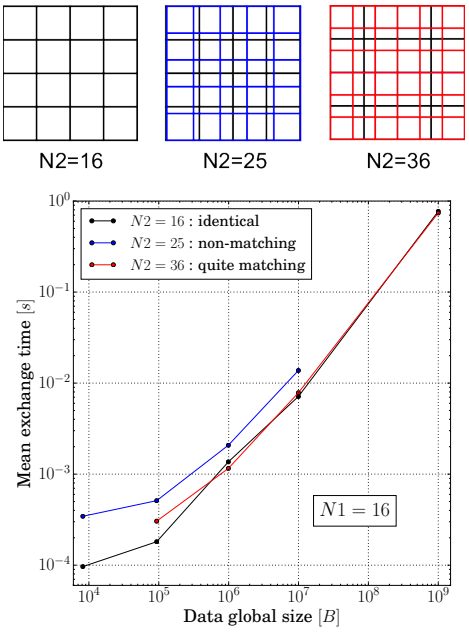


Fig. 7. Evolution of the exchange time as a function of the total amount of data on the grid, for cases where the partitioning of the two executables is either identical, or quite coincident, or totally non-coincident.

4 Conclusion

The CPU costs of a coupled simulation are determined both by the internal computational time of each code as well as by the interconnection process and the communication times between solvers. Core repartition between the coupled model to insure a good load balancing is rather trivial. Studying the effect of the data exchange time is much more complex and is examined here via a toy model. Many parameters can impact the communication times, such as the number of cores, the communication mode (synchronous or asynchronous), the global size of the exchanged fields or the amount of data per core. For the synchronous communications presented in this paper, two main outcomes independent from the coupler can be addressed by incorporating the knowledge of the coupling in the preprocessing step of the solvers with constraint and co-partitioning as well as process placement. Moreover, tests on asynchronous communications show an important improvement of the scalability of the coupler indicating development paths for the future. Finally, many order of magnitude higher than the communication time, the time requested by the interconnection process also depends on several parameters such as core distribution between the coupled components. Nevertheless, the real gain to decrease its CPU cost relies on interconnection algorithms and thus on further development in coupling libraries rather than on the global management of the coupling environment.

References

1. Berger, S.: Implementation of a coupled computational chain to the combustion chamber's heat transfer. Ph.D. thesis, Institut National Polytechnique de Toulouse (June 2016)
2. Buis, S., Piacentini, A., Déclat, D.: Palm: a computational framework for assembling high-performance computing applications. Concurrency and Computation: Practice and experience 18(2), 231–245 (2006)
3. Duchaine, F., Dombard, J., Gicquel, L., Koupper, C.: Integrated large-eddy simulation of combustion chamber / turbone interactiàons. In: 51st 3AF International Conference on Applied Aerodynamics. Strasbourg, France (4-6 April 2016)
4. Duchaine, F., Jauré, S., Poitou, D., Quémerais, E., Staffelbach, G., Morel, T., Gicquel, L.: Analysis of high performance conjugate heat transfer with the openpalm coupler. Journal of Computational Science and Discovery 8, 015003 (2015)
5. Duchaine, F., Maheu, N., Moureau, V., Balarac, G., Moreau., S.: Large eddy simulation and conjugate heat transfer around a low-mach turbine blade. J. Turbomach. 136(5) (2013)
6. Gicquel, L.Y.M., Staffelbach, G., Poinsot, T.: Large eddy simulations of gaseous flames in gas turbine combustion chambers. Prog. Energy Comb. Sci. 38(6), 782 – 817 (2012)
7. Gicquel, L., Gourdain, N., Boussuge, J.F., Deniau, H., Staffelbach, G., Wolf, P., Poinsot, T.: High performance parallel computing of flows in complex geometries. Comptes Rendus Mécanique 339(2-3), 104 – 124 (2011)
8. Jauré, S., Duchaine, F., Staffelbach, G., Gicquel, L.: Massively parallel conjugate heat transfer solver based on large eddy simulation and application to an aeronautical combustion chamber. Comput. Sci. Disc. Submitted (2013)

9. de Laborderie, J., Duchaine, F., Vermorel, O., Gicquel, L.: Application of an overset grid method to the large eddy simulation of a high-speed multistage axial compressor. In: ASME Turbo Expo 2016: Turbomachinery Technical Conference and Exposition. No. GT2016-56344, Seoul, Korea (13-17 June 2016)
10. N. Gourdain F. Sicot, F.D., Gicquel, L.: Large eddy simulation of flows in industrial compressors: a path from 2015 to 2035. Philosophical Transactions A 372(2022) (2014)
11. Piacentini, A., Morel, T., Thévenin, A., Duchaine, F.: O-palm: An open source dynamic parallel coupler. In: Proceedings of the IV International Conference on Computational Methods for Coupled Problems in Science and Engineering–Coupled Problems (2011)
12. Predari, M., Esnard, A.: Coupling-aware graph partitioning algorithms: Preliminary study. In: IEEE International Conference on High Performance Computing. Goa India (December 2014)
13. Refloch, A., Courbet, B., Murrone, A., Villedieu, P., Laurent, C., Gilbank, P., Troyes, J., Tessé, L., Chaineray, G., Dargaud, J., Quémerais, E., Vuillot, F.: Cfd platforms and coupling - cedre software. The Onera Journal Aerospace Lab (2) (2011)
14. Tucker, P., Eastwood, S., Klostermeier, C., Xia, H., Ray, P., Tyacke, J., Dawes, W.: Hybrid les approach for practical turbomachinery flows - part 2: further applications. J. Turbomach. 134(2) (2012)