In-Situ visualization in fluid mechanics using Catalyst: a case study for Code_Saturne

Benjamin Lorendeau, Yvan Fournier, and Alejandro Ribes *Member, IEEE*

Abstract— Numerical simulations using supercomputers are producing an increasingly larger volume of data to be visualized. In this context, Catalyst is a prototype In-Situ visualization library developed by Kitware to help reduce the data post-treatment overhead. On the other side, Code_Saturne is a Computational Fluid Dynamics code used at Eléctricité de France (EDF), one of the biggest electricity producers in Europe, for its large scale numerical simulations. In this article we present a study case where Catalyst is integrated into Code_Saturne. We evaluate the feasibility and performance of this integration by running two test cases in one of our corporate supercomputers.

Index Terms—In-Situ Visualization, Code_Saturne, Computational Fluid Dynamics, Large-Scale Scientific Simulations Visualization

1 Introduction

Computational Fluid Dynamics (CFD) is a fundamental step for the study and optimization of electricity production. Indeed, current power plants use water as a mean of convective heat transfer. Consequently, the simulation and visualization of fluid dynamics phenomena is of great importance for the energy industry. Electricité de France (EDF), being one of the biggest electricity producer in Europe, has developed for the past 15 years an open source CFD code named Code_Saturne, allowing for the solution of very large models [14]. EDF has several supercomputers that regularly run this code in order to perform analysis involving large amounts of data. In this context, the visualization of data becomes a critical point.

In the past, studies and improvements in scientific simulation have been mainly focused on the solver, due to being the most cycle-consuming part in the simulation process. Thus, visualization has been traditionally run sequentially on a smaller computer and at the very end of the solver computation. At the time, this was easily explained by the small need for both memory and computation resources in most of the visualization cases. Nevertheless, with the increase of our computational capabilities, we tend to use and generate much more data than what we were used to. Thus, as the scale of CFD simulation problems is getting wider, specific issues are emerging related to input/output efficiency. In particular, data generated during the solver computation and used for the visualization are the source of a worrisome overhead. Even worse, some researchers are starting to spend more time for writing and reading data than for running solvers and visualizations [12]. This new trend is asking us to design new I/O strategies and consider visualization as a part of our high-performance simulation systems.

Most fluid dynamic engineers at EDF R&D are currently visualizing lower temporal and spatial resolution versions of their simulations in order to avoid I/O issues when large quantities of data are involved. We decided to addres the subject of coprocessing and *in-situ* visualization which has been proved to be an effective solution against the current limitations of this problem [6], [13]. Our aim is to provide our engineers with an operational research-oriented tool in a mid-term basis. For this, we choose to evaluate Catalyst as an industrial tool for performing In-Situ visualization. Catalyst, developed by Kitware, is a library for Paraview that implements the coprocessing, by defining the visualization process through Paraview and exploiting the VTK's parallel algorithms for the processing of the simulation data [9].

- *Benjamin Lorendeau is M.Sc. student at University of Bordeaux 1. E-mail: benjamin.lorendeau@gmail.com.*
- *Yvan Fournier is with EDF R&D. E-mail: yvan.fournier@edf.fr.*
- *Alejandro Ribes is with EDF R&D. E-mail: alejandro.ribes@edf.fr.*

In this article, we propose a study upon the effectiveness and scalability of a prototype implementation of the coprocessing in an industrial case based on the coupling of Code_Saturne with Catalyst. In section 2 we discuss works related to recent visualization *in-situ* systems. We will then introduce in section 3 Code_Saturne, the CFD code developed at EDF R&D. In section 4 we present our integration of Catalyst into Code_Saturne and how it is used by the users in the framework of fluid dynamic simulations. Section 5 describes our use case and presents results on one of our corporate clusters. Finally, section 6 presents our analysis of the results and describes our on-going and future work.

2 Related Work

The size of generated data has become an important subject in high performance computing, due to the need of a better input/output efficiency in our computing system. To answer this problem, several visualization systems has been created. We can distinguish two main approaches in recent solutions. The first one is to integrate a specific *in-situ* visualization directly to our simulation code. Such an approach proved to be an efficient way to provide coprocessing for a given simulation plus visualization system as it is the case in the hurricane prediction [7] and earthquake [13] simulation systems. This method has been proved to lead to good performances but is limited to a specific implementation. Thus it does not respond to our needs.

The second approach is to provide a general postprocessing framework letting the simulation and the visualization code communicate together. EPSN which is a general coupling system, allows for the connection of M simulation nodes to N visualization nodes through a network [8]. This solution is a loosely coupled approach, requiring separate resources and data transfer through the network. This approach presents the advantage of not overloading the nodes used for computation. Thus the visualization code does not interfere with the execution of the simulation. Based on the same approach, a ParaView plugin named ICARUS [3] exists. It differs from EPSN in design by lower requirements as it only needs the use of a single HDF5 library and file driver extension. Such solutions offer tools for researchers to interact with their simulations by allowing them, first to monitor their current states but also to modify some parameters of the remaining simulation steps. Those computational steering solutions as well as the RealityGrid project [4] focus on interactivity with simulation whereas our main objective is to provide *in-situ* visualization operations to researchers while minimizing input/output overhead and disk space use.

Both built upon the well known parallel visualization algorithms library VTK, VisIt [5] and ParaView [10] provide through libsim [15] and Catalyst [9] the possibility to coprocess simulation data. Those *in-situ* solutions are tightly coupled and while they limit potential interactions with the running simulation, they also highly reduce the need of network data transfer. Thus, it contributes at circumventing the inefficiency of high performance computing input/output systems. Those solutions takes their benefits from directly accessing the simulation memory to perform visualization treatments by simply asking a pointer to the available data. One major drawback of this approach is the necessity to provide an understandable data layout to those libraries. Moreover, as this type of solution often gains from computing pre-determined visualization tasks, it is not suited for results exploration. As building a steering solution for Code_Saturne is out of the scope of this case study, we do not consider these drawbacks as a limitation.

After evaluating the performances offered by Kitware [9], we choose Catalyst as our coprocessing library for our case study as it answers our visualization needs while focusing on the reduce of data amount use. Ultimately, Kitware is still actively developing Catalyst, and we are optimistic that more services allowing the interactions with the running simulation will soon be available.

3 CODE_SATURNE: A COMPUTATIONAL FLUID DYNAMICS CODE

Code_Saturne is an open source computational fluid dynamics software designed to solve the Navier-Stokes equations in the cases of 2D, 2D axisymmetric or 3D flows. Its main module is designed for the simulation of flows which may be steady or unsteady, laminar or turbulent, incompressible or potentially dilatable, isothermal or not. Scalars and turbulent fluctuations of scalars can be taken into account. The code includes specific modules, referred to as "specific physics", for the treatment of lagrangian particle tracking, semi-transparent radiative transfer, gas combustion, pulverised coal combustion, electricity effects (Joule effect and electric arcs) and compressible flows. Code_Saturne relies on a finite volume discretisation and allows the use of various mesh types which may be hybrid (containing several kinds of elements) and may have structural non-conformities (hanging nodes).The parallelization is based on standard spatial decomposition with ghost cells that facilitate data passing between adjacent cells lying across the boundaries of disconnected decomposed parts using the Message Passing Interface. More technical details are presented in [1] and [2].

4 USING CATALYST

Catalyst is the general purpose coprocessing library of ParaView. This means that it was designed to work with any simulation code. This behavior is possible thanks to the use of an adaptor-based architecture. The adaptor binds the simulation code and Catalyst; it can access both the functions of the simulation code and the general-purpose API of Catalyst. As Catalyst itself is independent of the simulation code it is only the adaptor that should be implemented by the designers of the solver. In our case, we have developed a specific adaptor for Code_Saturne. Further explanations on the Catalyst design can be found in [9]. All our tests were run on "ParaView 3.98.0 enhanced" and "ParaView 3.98.1".

4.1 Implementing the adaptor

We identified two main issues to implement our adaptor, the memory management and the handling of ghost cell informations.

4.1.1 Memory management

In order to coprocess the simulation data, Catalyst must be provided with the data formatted to the VTK data object structure. To accomplish this, several solutions are possible, essentially depending on the format used for the data of the simulation code. In the case when the format of the simulation code is similar to VTK and, moreover, the simulation data can be shared at any time, then it is possible to feed Catalyst with a direct pointer to the simulation memory. Another option is to fully or partially copy the data from the simulation into a VTK object, and to send this object to Catalyst.

As we provide users of Code_Saturne with several output formats and as our data structure in the simulation differs from the VTK data object structure, feeding Catalyst with a direct pointer to the simulation memory is not possible. Thus, we chose to copy data from the simulation into a VTK data object. In fact, we are allocating vtkDoubleArray to store our data for Catalyst. Furthermore, we provide a pointer of those vtkDoubleArray to Code_Saturne so it can transform its simulation data and then fill the VTK data object.

The memory cost increase of our solution can be alleviated by using more machines. The cpu cost of the copy is in a range similar to the one needed when adapting simulation data to a specific output format. This cost is largely affordable comparatively to the time to write data to disk when storing time step specific outputs.

We are currently working on the evolution of our solution. Indeed Kitware plans to add specific *in-situ* data structures to VTK. This will offer adaptor developers facilities to make VTK access non VTK compatible simulation data.

4.1.2 Handling ghost cells: the vtkDistributedDataFilter

Handling ghost cell informations is also an important issue for us. Indeed, several Code_Saturne features and visualization filters in ParaView rely on the use of ghost data. Thus allowing the Code_Saturne and ParaView users to access this feature is an important objective in our industrial environment.

To address this need, we first tried to rely on the setting of nodes GlobalIds in order to see if ghost data exchanges between neighbors were already handled by Catalyst. Indeed ParaView implements a global numbering strategy on its nodes by using the so-called "GlobalIds". While the setting of these GlobalIds is relevant for the handling of ghost cell informations in ParaView, Catalyst does not actually use them. As we want Code_Saturne users to be able to use the larger scale of both simulation and visualization algorithms, we decided to force the application of the vtkDistributedDataFilter (D3) filter in the visualization pipeline. This D3 filter originally performs a redistribution of the data among the MPI processes but we use it to manage the ghost cells.

4.2 Pipeline Configuration Tools

From the point of view of an engineer performing a fluid mechanics simulation using Code_Saturne, the workflow of a coprocessing-simulation is: 1) to define a ParaView pipeline describing what the user wants to study and 2) to run the simulation. As users are already familiar with how to perform fluid mechanics simulations, defining the pipeline for the coprocessing will be their main issue. Thus this new process should be done in an efficient and easy way, at least this part should not become a cumbersome bottleneck. In our industrial environment this point is of great importance.

The definition of a Catalyst pipeline is possible in two ways. First, the pipeline can be defined programatically, a solution that we evaluate as too complicated and time consuming for the end user, especially when setting camera parameters is needed. Secondly, the pipeline can be created using the ParaView user's interface. This second solution is much easier for the user as he/she can simply interact with ParaView in the same way he/she used to when visualizing the results a posteriori. This last solution is the one we opted for and it can be performed after activating the coprocessing plugin in ParaView.

However, the chosen strategy for defining the pipeline implies a potential important bottleneck that we want to discuss in detail. Indeed, how can we a priori define a pipeline on the large geometry and fields that are going to be used in the simulation ? This is by itself a challenge and could imply a dedicated parallel system only to define the pipeline. Our solution consists in using a simplified or under-sampled version of the large geometry to define the pipeline. In fact, this strategy is possible in ParaView but some characteristics of the initial geometry must be present in its simplified version (principally equal names of fields).

Finally, the workflow of our engineers implies several steps to define the pipeline. First of all, the users possess a CAD (Computer Aided Design) version of the geometry that is parametrized. This parametric representation can generate meshes at different resolution

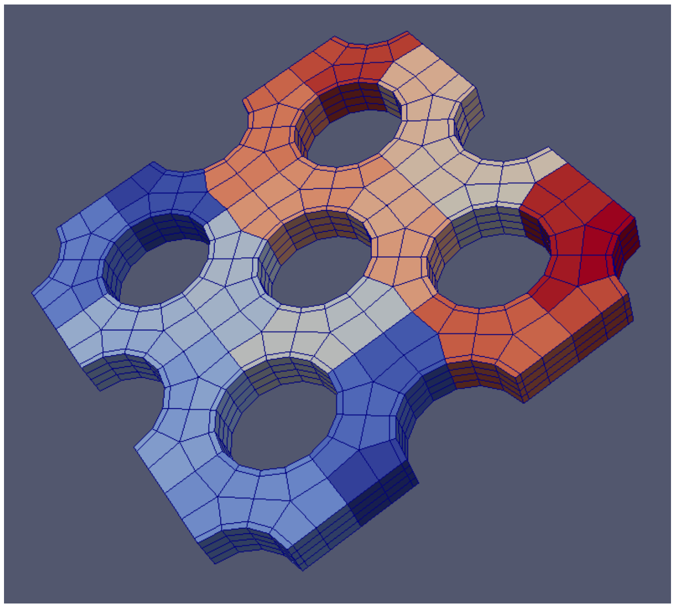
Fig. 1: Original geometry for our use case. The thickness is defined by layout and can be set to increase the complexity of the simulation. Here the number of layout is set to 8. In our test, the number of layout is set to 128.

factors. In our case, this is performed inside the open-source SALOME [11] platform for numerical simulation.

We generate two different meshes, one at high resolution (up to 204M hexahedrals in the use cases of this article) that will be used for the CFD simulation and one with a lower resolution to define the pipeline (700 000 hexahedrals in our use cases). The lowest resolution mesh is fed into Code_Saturne to perform a short simulation. This allows ParaView to obtain a representation containing not only the geometry but also the result fields. This is the data that is actually used to define the pipeline.

5 Results

5.1 Required User Interactions for Coprocessing

Before presenting our results we briefly describe how the user interactions was performed. The following steps were necessary in order to use the developed coprocessing technology:

1) The user generates a "light version" of the mesh. This step has already been discussed in section 4.2. Indeed, the user possess a CAD (Computer Aided Design) version of the geometry that is parametrized, it is then possible to obtain meshes at different spatial resolutions. A "light mesh" of small size in memory and representative of the CAD geometry is obtained. Figure 1 represents the "light version" of the mesh used in our experiments.

2) We run a short simulation (normally just a few seconds on a local machine) on the "light mesh". This obtains the informations about the result fields we need to create a visualisation pipeline in ParaView (e.g. temperature, pressure, velocity). We could then say that we obtain an "augmented light mesh".

3) The mesh and the fields obtained at the end of step 2 are read in ParaView and the user can define her/his visualisation pipeline. At the end of this step a simple click in the ParaView interface will create a Python file that programmatically defines the visualisation operations that will be performed *in-situ*.

4) Finally the real simulation is ran using a full resolution size mesh. The coprocessing part of the simulation reads the python script containing the definition of the visualisation pipeline. This step is expected to be time-consuming.

5.2 Use Cases

Our simulations have been run on Ivanoe, an EDF corporate supercomputer, composed of 1382 nodes, each node having 12 cores for a total of 16584 cores. In these simulations we associate one process by

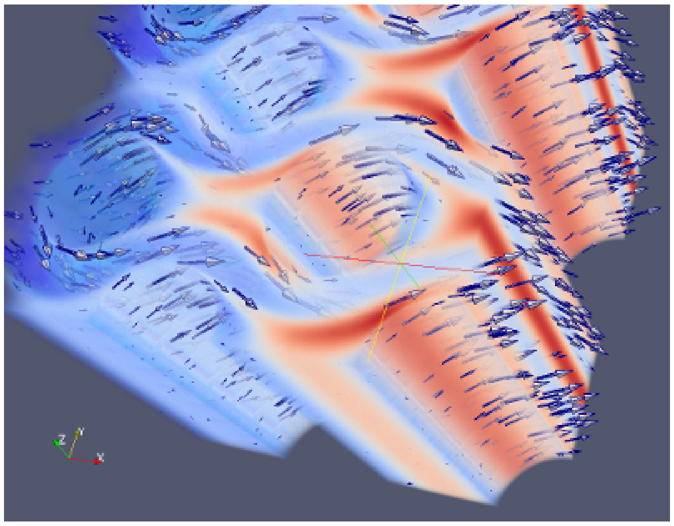

Fig. 2: A final coprocessed picture of our simulation. The pipeline where defined with a mesh of 8 layout while the simulation where run with 128 layout.

core and we use from 720 cores up to 3600 cores. We include two use cases that were run on this supercomputer. The choice of the cases is motivated by two main factors: the size of the mesh and the complexity of the visualization pipeline. Let us define in more detail why these two factors:

1) Mesh size. We chose to use two meshes representing the same geometry but at different resolutions, one made of 51M hexahedral elements and another of 204M hexahedrals. In our industrial environment at EDF most simulation engineers use meshes composed by less than 51M of element, then we choose this mesh size to be representative of the work performed by an average engineer in his work routine. Furthermore, a 51M elements mesh more than doubles the size used in the results presented in [9] for the PHASTA adaptor. On the other side, when researcher oriented simulations are performed at EDF, they currently contain around 200M elements. We chose then this size as a research oriented or 'heavy mesh' kind of simulation.

2) Pipeline complexity. We define two pipelines aimed to be representative of two different situations: users just performing simple and light visualization operations (mainly some slices in a volume) and another using very time-consuming visualization tasks (mainly performing a volume rendering).

In the following we name our uses cases: *CASE_A*, use case using an average mesh size of 51M hexahedrals and a visualization pipeline including volume rendering which aims to be very time-consuming. *CASE_B*, our second use case, contains a light visualization pipeline simply performing some slices but on a large mesh of 204M hexahedrals.

Table A summarizes the composition of these use cases. In all our use cases we run a simulation consisting in a fluid with physical properties identical to water passing through the mesh. Then the output is generated at each step, for a total of 10 coprocessed visualization images.

5.3 Results

Figure 2 presents an image obtained from one of our *in-situ* simulations with *CASE_A*. We see the flux of water moving around the vertical cylinders, the glyphs being attached to the velocity vectorial field. The color of the volume rendering represents the turbulent viscosity of the fluid. Figure 3 shows two graphs of *CASE_A*: in red the execution time versus the number of cores, in blue the execution time without the coprocessing overload. We are satisfied by this overload that is contained between 20 and 30% of the total execution time. It looks like this overload is reducing with the increase in number of cores. Figure 6 shows the exact same behavior but with *CASE_B*. Both graphs are difficult to distinguish as the time needed for coprocessing is

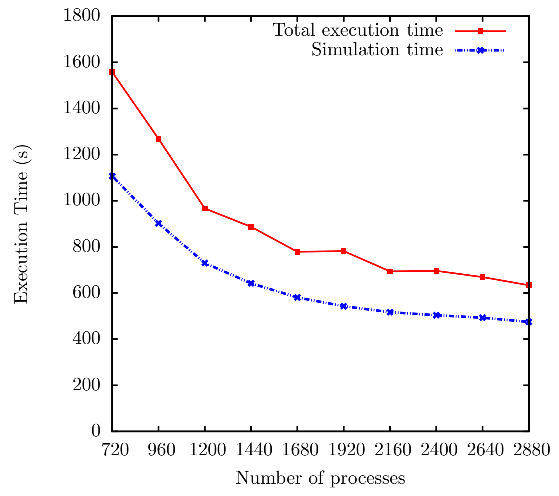

Fig. 3: CASE_A total execution time with and without the coprocessing.

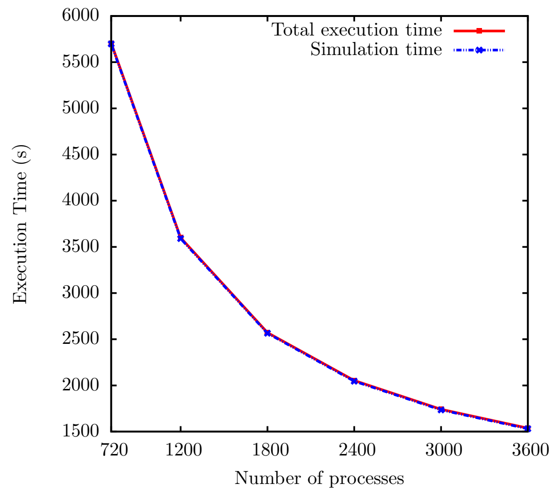

Fig. 6: CASE_B total execution time with and without the coprocessing.

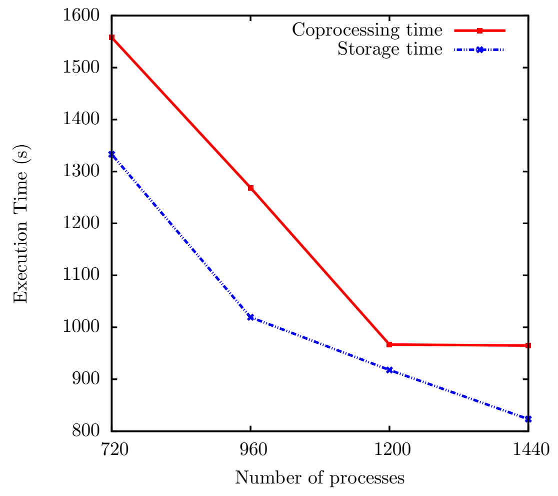

Fig. 4: CASE_A total execution time when using coprocessing and storage in Ensight Gold format.

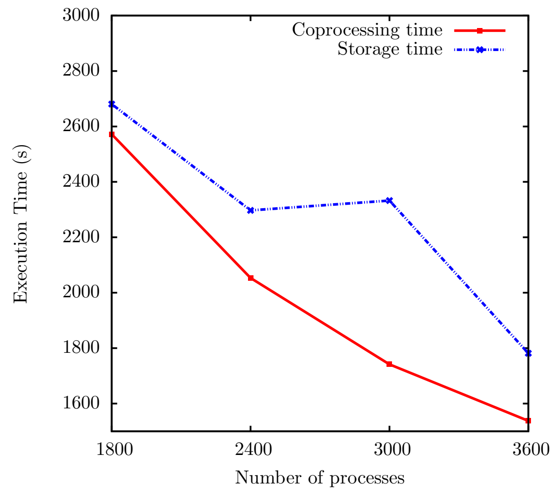

Fig. 7: CASE_B total execution time when using coprocessing and storage in Ensight Gold format.

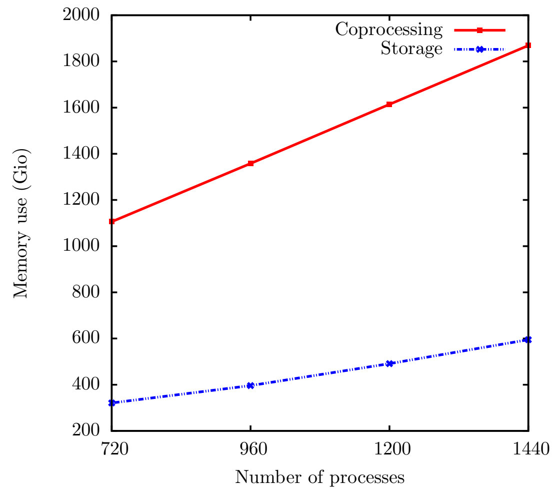

Fig. 5: CASE_A total memory usage when using coprocessing and storage in Ensight Gold format.

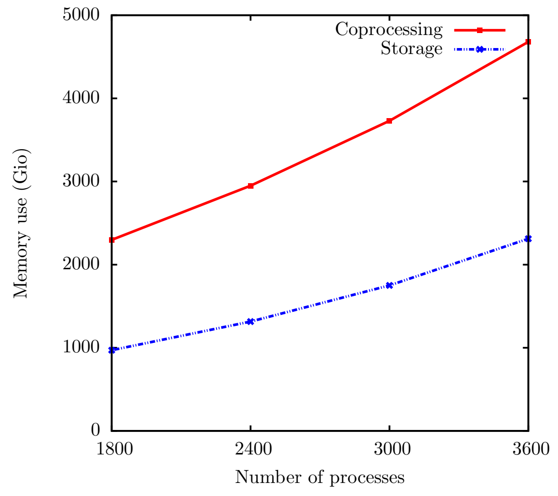

Fig. 8: CASE_B total memory usage when using coprocessing and storage in Ensight Gold format.

USE CASES SUM UP			
NAME	SIZE	PIPELINE	FIGURES
CASE_A	51M hexahedrals, industrial size case	**heavy**: volume rendering, celldatatopointdata and glyphs	3,4,5
CASE_B	204M hexahedrals, research size case	**light**: 9 slices, celldatatopointdata	6,7,8

Table A: Description of our two use cases.

circumscribed between 6 and 10 seconds, the overload is lesser than one percent of the total execution time.

We also decided to compare the Catalyst overhead with a non VTK based storage strategy that performs no visualization operations. Figure 4 and 7, show the comparison of the global execution time with Catalyst coprocessing versus the simple Ensight Gold format storage. Figure 4 presents our implementation results with *CASE_A*. This compares positively for Catalyst as the overhead is approximately 10% and looks decreasing when the number of cores increase. Figure 7 presents our results for *CASE_B*. Here we can see the potential of Catalyst when lighter and more relevant visualization tasks are processed. Indeed, there is no more overhead as we gain an average of 10% of execution time while freeing ourselves from storage issues (indeed, we evaluate the execution time peak of 3000 processes as a result of concurrent accesses on our supercomputer storage disks). To emphasize this, table B shows how much data each solution generates, namely a basic storage in Ensight Gold format versus our coprocessing implementation using Catalyst. These informations are those of our CASE_B when performing a 10 steps simulation. Both size are expected to grow proportionally to the size of the mesh input, and the number of steps. Therefore, we expect the gain provided by the use of coprocessing to be increasingly interesting when moving forward in use case size.

Finally, we also show the total memory use when running *in-situ* visualization compared to writing simulation results in Ensight Gold format in figure 5 and 8. We observe that memory consumption is increased by an approximate factor varying from 2 to 3. This can be explained by both our first naive memory management approach and also by a natural increase in memory consumption when visualization operations are to be performed. Indeed, our memory management implies a consumption increased by more than 2, as we need to translate data for Catalyst but still need the former data to pursue our simulation. Finally it may also be taken into account the actual memory consumption of the chosen visualization tasks.

*PROCESSING SIZE COMPARISON	
STORAGE	COPROCESSING
57Gio	1,3Mio

Table B: CASE_B comparison between the size of processed results and simple storage. The simulation was run on 10 steps, with 10 pictures coprocessed.

6 Conclusion

We have successfully integrated Catalyst into Code_Saturne (a computational fluid dynamics code developed at EDF R&D). After testing the prototype in our corporate supercomputer Ivanoe, we find Catalyst to be a relevant solution to provide Code_Saturne users with visualization coprocessing. Catalyst proved to allow a simple and fast implementation of an adaptor. We use D3 (a filter originally performing a redistribution of the data among MPI processes) as a ghost cell handler. We feel that the code responsible for the ghost cells management in D3 could be integrated directly into ParaView/Catalyst since applying D3 can be time consuming.

Our preliminary results are based on a 51M and a 204M elements mesh, which is above the average size case used by EDF engineers in our industrial environment. We plan to perform simulation on at least 400M elements meshes in the near future. We have also performed simulation up to 300 nodes and are currently planning not using more in this cluster. This is due to the typical simulation node size being around 150 nodes for our engineers. We also plan to work on another of our corporate supercomputers, an IBM BG/Q with 65k cores. In that case, we will test on a much bigger number of cores. The increase of memory use found in the results section indicates that memory optimizations are to be performed before running on the IBM BG/Q. We did not in this study perform any delicate memory tweaking in order to reduce the memory consumption. We are currently working on this point, experimenting new VTK *in-situ* data structures that may highly reduce this overhead.

We are mostly satisfied with the integration of Catalyst in Code_Saturne. Our first version of our integration will be most probably released in September 2013 as part of a new version of this opensource software.

References

[1] F. Archambeau, N. Mechitoua, and M. Sakiz. Code saturne: A finite volume code for the computation of turbulent incompressible flows. In *Industrial Applications, International Journal on Finite Volumes, Vol. 1*.

[2] F. Archambeau, N. Mechitoua, and M. Sakiz. Edf, code_saturne version 3.0 practical user's guide, 2013.

[3] J. Biddiscombe, J. Soumagne, G. Oger, D. Guibert, and J.-G. Piccinali. Parallel computational steering for hpc applications using hdf5 files in distributed shared memory. *Visualization and Computer Graphics, IEEE Transactions on*, 18(6):852–864, 2012.

[4] J. Brooke, P. Coveney, J. Harting, S. Pickles, Pinning, and A. R.L. Porter. Computational steering in realitygrid. *Proc. UK E-Science All Hands Meeting*, pages 885–888, 2003.

[5] H. Childs, E. Brugger, K. Bonnell, J. Meredith, M. Miller, B. Whitlock, and N. Max. A contract based system for large data visualization. In *Visualization, 2005. VIS 05. IEEE*, pages 191–198, 2005.

[6] K. D. M. David Thompson, Nathan D. Fabian and L. G. Ice. Design issues for performing in situ analysis of simulation data. In *Technical Report SAND2009-2014, Sandia National Laboratories*, pages 7–18, 2009.

[7] D. Ellsworth, B. Green, C. Henze, P. Moran, and T. Sandstrom. Concurrent visualization in a production supercomputing environment. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):997–1004, 2006.

[8] A. Esnard, N. Richart, and O. Coulaud. A steering environment for online parallel visualization of legacy parallel simulations. In *Distributed Simulation and Real-Time Applications, 2006. DS-RT'06. Tenth IEEE International Symposium on*, pages 7–14, 2006.

[9] N. Fabian, K. Moreland, D. Thompson, A. Bauer, P. Marion, B. Geveci, M. Rasquin, and K. Jansen. The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 89–96, 2011.

[10] C. Law, A. Henderson, and J. Ahrens. An application architecture for large data visualization: a case study. In *Parallel and Large-Data Visualization and Graphics, 2001. Proceedings. IEEE 2001 Symposium on*, pages 125–159, 2001.

[11] A. Ribes and C. Caremoli. Salome platform component model for numerical simulation. In *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, volume 2, pages 553–564, 2007.

[12] R. B. Ross, T. Peterka, H.-W. Shen, Y. Hong, K.-L. Ma, H. Yu, and K. Moreland. Visualization and parallel i/o at extreme scale. *Journal of Physics: Conference Series*, 125(1):012099, 2008.

[13] T. Tu, H. Yu, L. Ramirez-Guzman, J. Bielak, O. Ghattas, K.-L. Ma, and D. O'Hallaron. From mesh generation to scientific visualization: An end-to-end approach to parallel supercomputing. In *SC 2006 Conference, Proceedings of the ACM/IEEE*, pages 12–12, 2006.

[14] P. Vezolle, J. Heyman, B. D'Amora, G. Braudaway, K. Magerlein, J. Magerlein, and Y. Fournier. Accelerating computational fluid dynamics on the ibm blue gene/p supercomputer. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2010 22nd International Symposium on*, pages 159–166, 2010.

[15] B. Whitlock, J. M. Favre, and J. S. Meredith. Parallel in situ coupling of simulation with a fully featured visualization system. In *Proceedings of the 11th Eurographics conference on Parallel Graphics and Visualization*, EG PGV'11, pages 101–109, Aire-la-Ville, Switzerland, Switzerland, 2011. Eurographics Association.