

In Situ Statistical Analysis for Parametric Studies

Théophile Terraz
Univ. Grenoble Alpes, INRIA
France

Bruno Raffin
Univ. Grenoble Alpes, INRIA
France

Alejandro Ribes
EDF Lab Paris-Saclay
France

Yvan Fournier
EDF Lab Paris-Chatou
France

Abstract—In situ processing proposes to reduce storage needs and I/O traffic by processing results of parallel simulations as soon as they are available in the memory of the compute processes. We focus here on computing in situ statistics on the results of N simulations from a *parametric study*. The classical approach consists in running various instances of the same simulation with different values of input parameters. Results are then saved to disks and statistics are computed post mortem, leading to very I/O intensive applications. Our solution is to develop Melissa, an in situ library running on staging nodes as a parallel server. When starting, simulations connect to Melissa and send the results of each time step to Melissa as soon as they are available. Melissa implements iterative versions of classical statistical operations, enabling to update results as soon as a new time step from a simulation is available. Once all statistics are updated, the time step can be discarded. We also discuss two different approaches for scheduling simulation runs: the jobs-in-job and the multi-jobs approaches. Experiments run instances of the Computational Fluid Dynamics Open Source solver *Code_Saturne*. They confirm that our approach enables one to avoid storing simulation results to disk or in memory.

Index Terms—Parallel processing; Data processing

I. INTRODUCTION

Large scale simulations are producing an ever growing amount of data that are overloading the machine I/Os, impacting the performance of both the simulation when saving the data, and the post hoc analysis when reading them. *In situ processing* proposes to move away from the standard approach that consists in saving raw data to disks and then perform result analysis post mortem. In situ aims at reducing data traffic and speeding-up result analysis by performing result processing (compression, indexation, analysis, visualization, etc.) as closely as possible to the locus and time of data generation [1]. Research to make in situ processing efficient often focuses on how to perform analytic on the data produced by a single large parallel simulation [2]. Investigated issues include the development of various frameworks for easing coupling the simulation and the analysis [3], [4], [5], advanced strategies for resource sharing between the simulation and the analysis [6], as well as new algorithms taking into account the specific balance of the in situ processing context (need to save on data movements) [7].

In this paper we consider a different context where we compute statistics combining the results of N simulations to perform a *parametric study*. The result of such a family of runs is called *ensemble data set*, and each individual run is called a *member*. Ensembles are also multidimensional, multivariate

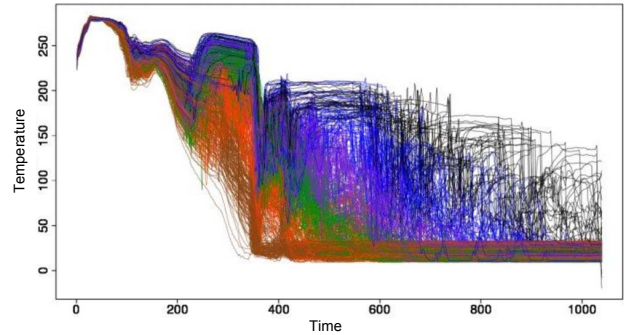


Figure 1: Visualization of 600 Monte Carlo simulations of thermo-hydraulical transients.

and multivariate [8]. Challenges in analysing and visualizing ensembles stem from the size and complexity of the data [9].

The classical approach consists in running various instances of the same simulation with different values for some input parameters. Results are saved to disks and statistics computed post mortem. The base scenario is to compute statistics, an average for instance, over the various values that take a given parameter for a given mesh element at a given time (x, y, z, t) through the N simulations. To avoid being overwhelmed with data, often the simulation outputs the data for a sub-sample of all the computed (x, y, z, t) points.

Parametric studies are becoming increasingly popular in industrial environments. For instance a consortium of companies, including EDF, one of the biggest electricity producer in the world, has developed specific software infrastructures for parametric studies like OpenTurns [10]. But such environments are working with data stored in files and cannot cope with very large amount of data. A solver like *Code_Saturne*, a CFD code developed at EDF, can run large scale numerical simulations [11], dealing with meshes from several million up to a few billion million cells. *Code_Saturne* already supports in-situ visualisation based on Catalyst [12], but for a single run. Performing analysis on several runs requires to an amount of data that is several order of magnitude larger. With the hypothesis that the solver is (using the nomenclature of [8]):

- multidimensional: we consider a 3D mesh with 1,000 million cells and 200 time steps,
- multivariate: we consider calculating 10 result vector fields (with 3 components per field element),

- multivalued: we consider a simple parametric study with $N=100$ runs,

the ensemble size would reach 1,200 TB ($10^9 \times 200 \times 10 \times 3 \times 100 \times 2$ considering data coded on 2 bytes) or 12 TB per simulation run. One could argue that such 1,000 million cells mesh has never been used in parametric studies before. If we consider a simpler 100 times smaller mesh of 10 million cells, a scale today common in industrial environments, we would still get an ensemble of 12 TB. To reduce the data size, a first solution commonly adopted in industrial parametric studies consists in eliminating part of the complexity by choosing one variable (eliminate multivariability) and one point in the simulation (eliminate spatial multidimensionality). This solution leads to a reduced dataset where only the multiple values of one parameter for the different simulations and timesteps are kept. Fig. 1 shows 600 Monte Carlo simulations of temperature in function of time in a thermo-hydraulical transient parametrical simulation performed at EDF, simulating a large break loss of primary coolant accident in a power plant. The engineer can study the loss of temperature by exploring this set of curves but they are focusing on a specific variable and on a specific location in the simulation domain. Attempting to compute statistics for all variables and locations handling data stored on disks would be extremely time-consuming or just not possible today.

In this paper we present early work to compute efficiently the statistics for all variables, locations and timesteps, by relying on an in situ processing approach. The goal is to avoid having to save the simulation results to disk, computing the statistics as soon as available from the simulations. Because we bypass the disk, we can envision to compute the statistics at a high sampling rate, or, what we considered for our experiments, at full resolution. For that we have developed a library called Melissa (Modular External Library for In Situ Statistical Analysis), running on staging nodes as a parallel server. When starting, simulations connect to Melissa and forward it the results of each time step as soon as available. Melissa implements iterative versions of classical statistic operations (average, standard deviation, minimum, maximum and threshold exceedance) enabling to update results as soon as a new time step from a simulation is available. Once all statistics updated, the time step can be discarded. We also discuss two different approaches for scheduling simulation runs. Each run can be submitted to the machine batch scheduler, letting the scheduler optimize resource allocation globally in between all submitted jobs from the different users. The second approach consists in requesting once the necessary resources for running all simulations as well as Melissa, and then use our own scheduling strategy on the allocated resources. After presenting Melissa Architecture (Sec. II), we introduce some early experiments (Sec. III).

II. ARCHITECTURE

A. Iterative Statistics

We consider a numerical simulation on a fixed mesh. The simulation computes values for different fields u for each mesh

cell or node. As meshes have usually 3 dimensions, we denote by $X = (x, y, z)$ each node or cell. The simulation progresses in time through various time steps (all simulations simulate the same time steps). Let t be the time step index. The same simulation runs N times with different input parameters. Let call i the i^{th} simulation. The goal is to compute statistics over all simulations for each mesh element at each time step and for each field $u(i, X, t)$. For instance, the classical formula to compute a simple mean of the field u for each (X, t) over the N simulations is:

$$\mu(X, t) = \frac{\sum_{i=1, N} u(i, X, t)}{N}$$

Our goal is to compute such statistics in situ with a minimal memory footprint. If the statistics can be *computed iteratively*, i.e. if the current value can be updated as soon as incoming results are available, we would not need to save the simulation results. This is actually the case for the simple mean μ that can be formulated iteratively:

$$\mu_i(X, t) = \mu_{i-1}(X, t) + \frac{1}{i}(u(i, X, t) - \mu_{i-1}(X, t))$$

with $\mu_0(X, t) = 0$ and $1 \leq i \leq N$. Similarly for the variance:

$$V_i(X, t) = V_{i-1}(X, t) + (u(i, X, t) - \mu_{i-1}(X, t))(u(i, X, t) - \mu_i(X, t))$$

with $V_0(X, t) = 0$ and $1 \leq i \leq N$. Not only simulation results do not need to be saved, but they can be consumed in any order, loosening synchronisation constraints on the simulation executions. More generally, a given statistic *Stat* can be computed iteratively if it can be written as:

$$Stat_i(X, t) = f(Stat_{i-1}(X, t), u(i, X, t)).$$

We implemented iterative statistics for the simple mean, variance, minimum and maximum, threshold exceedance following [13]. In all cases, simulation results can be processed in any order. It remains to be seen if all statistics that a user may need, can be formulated iteratively.

B. In Transit Data Processing

Given that the statistics we perform combine the results obtained from different simulations on the same mesh element and time step, we based Melissa architecture on an in transit processing model where simulations results are forwarded as soon as available to the staging nodes Melissa runs on (Fig. 2). Combining an in situ approach with iterative statistics enables to drastically reduce the amount of memory needed. No simulation is saved to disk. Melissa only needs to keep in memory the current version of each statistic computed. It is of the order of the size of the results of one simulation. Increasing the number of staging nodes enables to increase the amount of memory available. Memory size can also be extended by relying on out-of-core memory (using local SSD disks or burst buffers for instance) but Melissa does not support it yet.

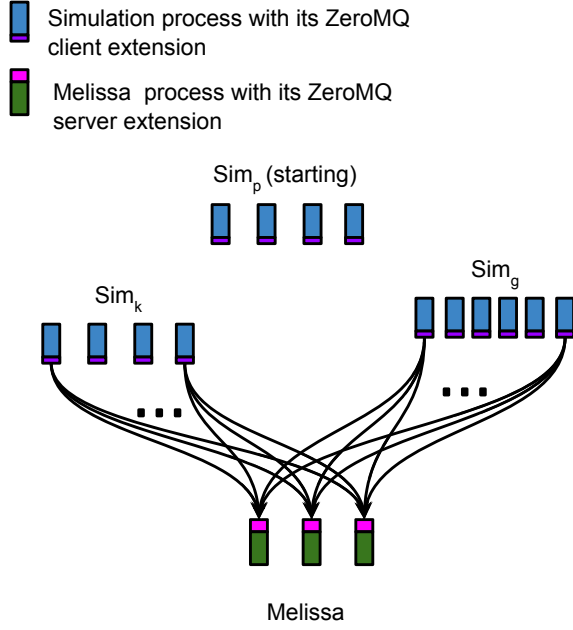


Figure 2: Melissa architecture diagram. Simulations connect to Melissa when starting and forward results as soon as available.

C. Client/Server

Melissa relies on a client/server model. This is not classical in HPC environments, but it is a flexible approach that fits well the particular execution scheme of parametric studies. A Melissa server runs in parallel several MPI processes that can be distributed on different nodes. Melissa runs as a service waiting for simulations to connect and forward their results. Once the connection between one simulation and Melissa is established, each process from the simulation directly distributes its results to the target Melissa processes that need them, thus limiting data copies. Messages received are directly processed to update the computed statistics. Melissa can process the incoming messages from several simulations running simultaneously. Melissa adopts its own partition of the simulation space that is so far defined statically at start time. On the simulation side, the modifications required are minimal. The code needs to embed the logic for establishing the connection with Melissa, the routine to scatter the data to the correct target Melissa processes, and connection closing.

Current Melissa implementation relies on the ZEROMQ communication library [14] commonly used for distributed applications. ZEROMQ allows several clients to connect to the same server port, and takes care of message transfer, buffering and aggregation in the background. It proved very convenient for development. The main ZEROMQ limitation we identified so far is the lack of direct support for high performance networks like Infiniband. It needs to rely on IP over infiniband instead. However in our context we did not experience any performance issue, neither on the simulation side nor on the server side. We did attempt to use MPI_connect, but it proved

more cumbersome than ZEROMQ, leading to a more complex code without any visible performance benefit.

We could rely on a solution like DataSpaces [5] that provides in-memory data staging for parallel simulations. But we actually do not need to store the simulation results as they are consumed as soon as produced.

D. Resource Allocation

The client/server model allows to run each simulation independently in different execution contexts. This flexibility enables to envision two different ways for scheduling executions:

- **Jobs-in-job:** the user first request one set of processors for executing all the simulation runs and one instance of Melissa to the machine scheduler in charge of deciding when and where to execute submitted jobs. Then, this is the user responsibility to schedule the different jobs within the envelop of resources allocated by the machine scheduler. One option is to rely on a traditional batch scheduler. Some like OAR [15] support such scheduling scheme. Once the resources available, the user can have a good estimate of the full execution duration.
- **Multi-jobs:** each simulation run is submitted individually to the machine batch scheduler. The only constraint is to make sure that Melissa is running from the first to the last simulation execution, and that the communication between jobs from the same user are allowed on the machine. Exposing all the runs to the machine batch scheduler allows it to take benefit of their independence to better leverage the machine resources. But the global duration of the experiment is more uncertain compared to the previous approach.

Notice that because Melissa does not need all the runs to execute with the same number of processes, the number of processors allocated to each run could be decided by the batch scheduler. Though it has been shown that scheduling these *moldable jobs* is more efficient than having rigid jobs, this is a feature supported by some batch schedulers but that we did not test yet in our context [16].

III. EXPERIMENTS

We present here a first set of early experiments at modest scale. All the computations presented in this paper were performed on the Froggy machine. Each Froggy node have two Intel E5-2670 (Sandybridge) 2.6 Ghz 8 cores processors, 64 GB memory, and are interconnected by a FDR InfiniBand network. The batch scheduler used on this machine is OAR¹.

The simulation code is *Code_Saturne*², a parallel CFD code developed by Electricité de France (EDF). The statistical analysis is performed by Melissa. The use case created by EDF for this experiment simulates a purge of a volume of hot water by introducing a flow of cold water (Fig. 3). The parametric study consists in varying the initial temperature

¹<https://oar.imag.fr/>

²<http://code-saturne.org/>

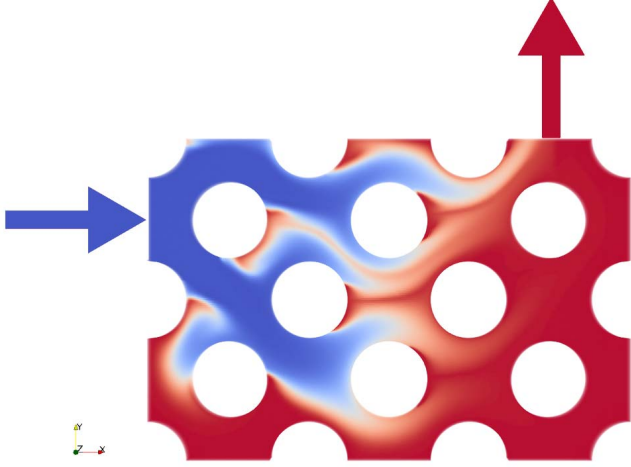


Figure 3: The use case : a cavity filled with 300°C water, chased by a stream of 25°C water.

of the hot water from 300°C to 350°C by 5°C steps, and the temperature of the cold water from 20°C to 30°C by 1°C steps. For this experiment, Melissa computes the mean, variance and min and max of temperature and pressure for each element of the mesh at each time step. We used a 3D mesh of 6 002 400 elements. Adapting *Code_Saturne* to support Melissa was straightforward. We wrote a new *writer* (*Code_Saturne* plugin that encapsulates the output data management code) based on Melissa. This writer code is about 400 lines of codes including 40 Melissa specific ones.

We first ran one simulation writing the outputs to disk (Ensign Gold data format). The output is 24 MB per time step and per output field. If we run 100 simulations with 30 time steps and two output fields each, this leads to write 144 GB on disc. When analysing the results in situ these data are directly processed by Melissa. Disk usage is limited to storing the final statistics (24 MB per time step per field and per statistic, for a total of 5.76 GB). Statistics for each time step can next be visualized with traditional tools like VTK/Paraview (Fig. 4).

Experiments also confirm that computing the statistics is a lightweight work compared to the simulation. Running Melissa on 16 processes (one node) and the simulation on up to 128 cores (8 nodes), Melissa processes the data produced by one time step in less than 0.02% of the time it takes to compute these results. When running 100 simulations in parallel, each one on 16 processes (1 node), Melissa load is still below 1%.

We tested the two different scheduling strategies presented in II-D using the OAR batch scheduler. The jobs-in-job scheduling can be performed with OAR by first submitting a specific empty job called a *container* to request for the allocation of a set of resources. Then, we submit independently the simulations and Melissa job as usual, but requesting OAR to schedule these jobs in the container only. As expected on an empty cluster, the delay between the first job submission and the completion of the executions was similar for both cases (multi-jobs and jobs-in-job). On a busy cluster, launching

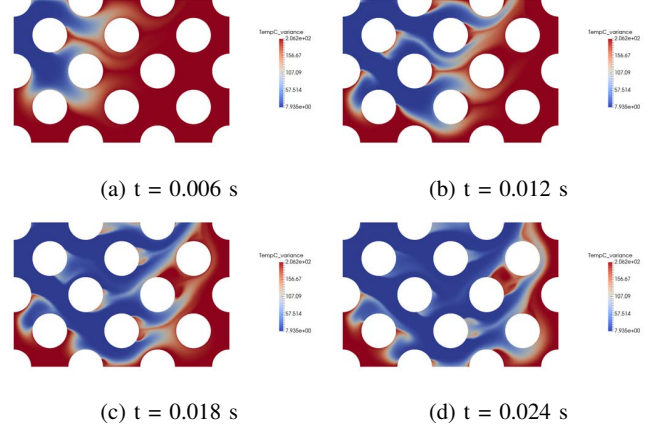


Figure 4: Heat variance fields for 100 simulations at four different time steps on an horizontal slice

multi-jobs leads to a faster scheduling.

IV. CONCLUSION

We presented the early development of Melissa, a library for computing statistics for large scale parametric studies. Melissa runs as a parallel server on staging nodes, processing the results of each time step as soon as available. Melissa enables to bypass the storage of simulation results to disk and requires memory to store the computed statistics only. Such gains rely on the capability to compute the statistics iteratively. Future work include large scale experiments, adding more statistic operations as well as extending Melissa with fault tolerance mechanisms. Melissa is still at an early development stage. We expect to open source the code and make it available in the coming months.

ACKNOWLEDGEMENT

This work was partly funded in the Programme d'Investissements d'Avenir, grant PIA-FSN2 -Calcul intensif et simulation numérique - 2 - AVIDO. All presented computations were performed on the CIMENT infrastructure³, which is supported by the Rhône-Alpes region (GRANT CPER07_13 CIRA) and the Equip@Meso project (reference ANR-10-EQPX-29-01) of the programme Investissements d'Avenir supervised by the Agence Nationale pour la Recherche.

REFERENCES

- [1] H. Yu, C. Wang, R. Grout, J. Chen, and K.-L. Ma, "In situ visualization for large-scale combustion simulations," *Computer Graphics and Applications, IEEE*, vol. 30, no. 3, pp. 45–57, 2010.
- [2] M. Dorier, M. Dreher, T. Peterka, J. M. Wozniak, G. Antoniu, and B. Raffin, "Lessons Learned from Building In Situ Coupling Frameworks," in *Workshop on In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization (ISAV'15)- Held in conjunction with SC15*. Austin: ACM, Nov. 2015.

³<https://ciment.ujf-grenoble.fr>

- [3] M. Dorier, G. Antoniu, F. Cappello, M. Snir, and L. Orf, "Damaris: How to Efficiently Leverage Multicore Parallelism to Achieve Scalable, Jitter-free I/O," in *CLUSTER - IEEE International Conference on Cluster Computing*. IEEE, Sep. 2012.
- [4] M. Dreher and B. Raffin, "A Flexible Framework for Asynchronous In Situ and In Transit Analytics for Scientific Simulations," in *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. Chicago, United States: IEEE Computer Science Press, May 2014. [Online]. Available: <https://hal.inria.fr/hal-00941413>
- [5] C. Docan, M. Parashar, and S. Klasky, "DataSpaces: an Interaction and Coordination Framework for Coupled Simulation Workflows," *Cluster Computing*, vol. 15, no. 2, pp. 163–181, 2012.
- [6] F. Zheng, H. Yu, C. Hantas, M. Wolf, G. Eisenhauer, K. Schwan, H. Abbasi, and S. Klasky, "GoldRush: Resource Efficient in Situ Scientific Data Analytics Using Fine-grained Interference Aware Execution," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 78:1–78:12. [Online]. Available: <http://doi.acm.org/10.1145/2503210.2503279>
- [7] A. Agranovsky, D. Camp, C. Garth, E. Bethel, K. Joy, and H. Childs, "Improved post hoc flow analysis via lagrangian representations," in *Large Data Analysis and Visualization (LDAV), 2014 IEEE 4th Symposium on*, Nov 2014, pp. 67–75.
- [8] A. L. Love, A. Pang, and D. L. Kao, "Visualizing spatial multivalued data," *IEEE Computer Graphics and Applications*, vol. 25, no. 3, pp. 69–79, May 2005.
- [9] A. T. Wilson and K. C. Potter, "Toward visual analysis of ensemble data sets," in *Proceedings of the 2009 Workshop on Ultrascale Visualization*, ser. UltraVis '09. New York, NY, USA: ACM, 2009, pp. 48–53. [Online]. Available: <http://doi.acm.org/10.1145/1838544.1838551>
- [10] EDF, E. France, and P. Engineering, "Openturns, <http://www.openturns.org>."
- [11] Y. Fournier, J. Bonelle, C. Moulinec, Z. Shang, A. Sunderland, and J. Uribe, "Optimizing code_saturne computations on petascale systems," *Computers & Fluids*, vol. 45, no. 1, pp. 103 – 108, 2011, 22nd International Conference on Parallel Computational Fluid Dynamics (ParCFD 2010)ParCFD. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0045793011000351>
- [12] benjamin Lorendeau, Y. Fournier, and A. Ribes, "In Situ visualization in fluid mechanics using Catalyst: a case study for Code_Saturne," in *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, 2013.
- [13] T. Finch, "Incremental calculation of weighted mean and variance," University of Cambridge, Tech. Rep., 2009.
- [14] P. Hintjens, *ZeroMQ, Messaging for Many Applications*. O'Reilly Media, 2013.
- [15] N. Capit, G. D. Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounié, P. Neyron, and O. Richard, "A batch scheduler with high level components," in *5th International Symposium on Cluster Computing and the Grid (CCGrid)*, 2005.
- [16] M. C. Cera, Y. Georgiou, O. Richard, N. Maillard, and P. O. A. Navaux, "Supporting MPI Malleable Applications upon the OAR Resource Manager," in *Colibri : Colloque d'Informatique: Brésil / INRIA, Coopérations, Avancées et Défis*, Rio Grande do Sul, Brazil, Jun. 2009. [Online]. Available: <https://hal.inria.fr/hal-00691414>