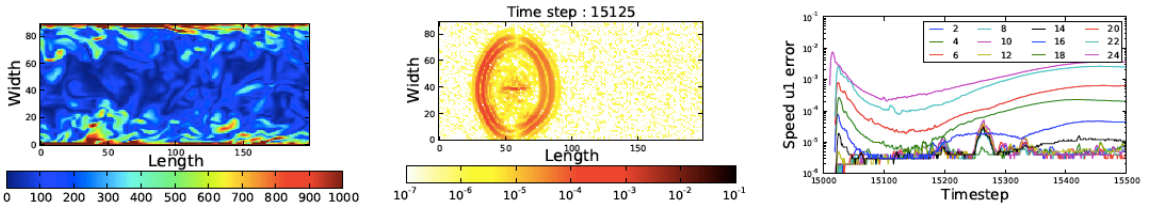
# Using data analytics to detect corruptions in numerical simulations on potentially unreliable environments

Franck Cappello, Leonardo Bautista Gomez, Sheng Di, ANL

As we progress toward executions of numerical simulations at the exascale, we are realizing that these executions may face an unprecedented risk of corruptions coming from diverse sources. Recent news concerning a leadership system using GPUs reminds us that corruptions can come from bugs on some hardware components. Other examples of corruptions come from the software. Indeed, the complexity involved in running large-scale numerical simulations has reached a level that the correctness of numerical simulation executions cannot be assumed based on the belief that the execution environment (the application, software, and hardware) is trustworthy. More generally, not only do we need to worry about corruptions that may come from radiation, but we also need to consider the risk that any part of the hardware or software involved in the execution is a potential source of corruption.

This problem is related to the notions of trust in simulation results and scientific computing integrity.

Figure 1 shows the simulation of a turbulent fluid (a), the propagation of a corruption (bit 24) at point 40x40 of the velocity field, 125 time steps after injection (b), and the maximum deviations after a corruption at different bit positions (2, 4, 6, etc.) (c).




(a) Vorticity in turbulent fluid (b) Error propagation of 24th bit corruption (c) Maximum deviation after corruption

Fig. 1: Simulation of turbulent fluid and propagation of data corruption

A classic avenue to address this problem is to develop specific tests that will detect corruptions. Most simulation applications have sanity checks verifying invariants and the preservation of some properties (for example, checking that pressures in ocean are all positive). Simple tests are often not precise enough, however, to catch corruptions that introduce errors at the scale of accuracy expected by users (typically $10^{-6}$–$10^{-8}$).

If we consider that corruptions can come from any hardware or software component, we cannot rely on replication as a general solution for this problem. Although replication does help detect nonsystematic errors (typically, radiation-induced silent data corruptions), it cannot help detect systematic errors (such as bugs in system software or hardware).

We also need to deal with another issue. In scenarios where the application itself is the source of corruptions, we cannot rely only on detectors that the programmer will have embedded in the application, because they may be deficient, too.

The domain of highly reliable systems has already faced this situation, and solutions such as N-version programming [1] and recovery block [2] compare the results of multiple versions (or alternatives) of the same hardware or software component. The objective is to increase the diversity in order to avoid common errors. This approach assumes that the different versions follow the same specifications, which usually translate into the same or similar complexities for

all versions. But such an approach is difficult to adapt to exascale numerical simulations, for at least two reasons: (1) exascale applications are very complex codes, and producing multiple versions of these codes would be expensive; and (2) multiversions is a form of replication, and replication is considered too expensive in resources and energy in the context of exascale numerical simulations.

To detect corruptions at the scale of accuracy expected by users with a lower hardware/software complexity than the multiple versions approach, we need to develop new, low-overhead detection techniques that focus on end-user needs.

To this end, we explore corruption detectors based on data analytics for applications where the state variables of a physical system (position, velocity, energy, pressure, temperature, etc.) are computed for multiple time steps. These applications often use a complex sequence of operations, including calls to solvers and fast Fourier transforms, to compute the transformation of the variables for each time step. Our approach [3] monitors these transformations. For several applications run with production data sets, we have observed that these transformations are spatially and temporally smooth enough (for each variable taken individually) to be checked by simple tests based on data clustering and regression. We do not claim that all simulation variables have smooth trajectories; but for those that do have this property, simple on-line tests provide surprisingly good corruption detection. Figure 2 shows the detection capability of five variable-tracking algorithms with respect to corruptions introduced in the different bit positions of a simulation variable for four applications: the HACC cosmology code, a turbulence CFD kernel, Nek5000, and FLASH.

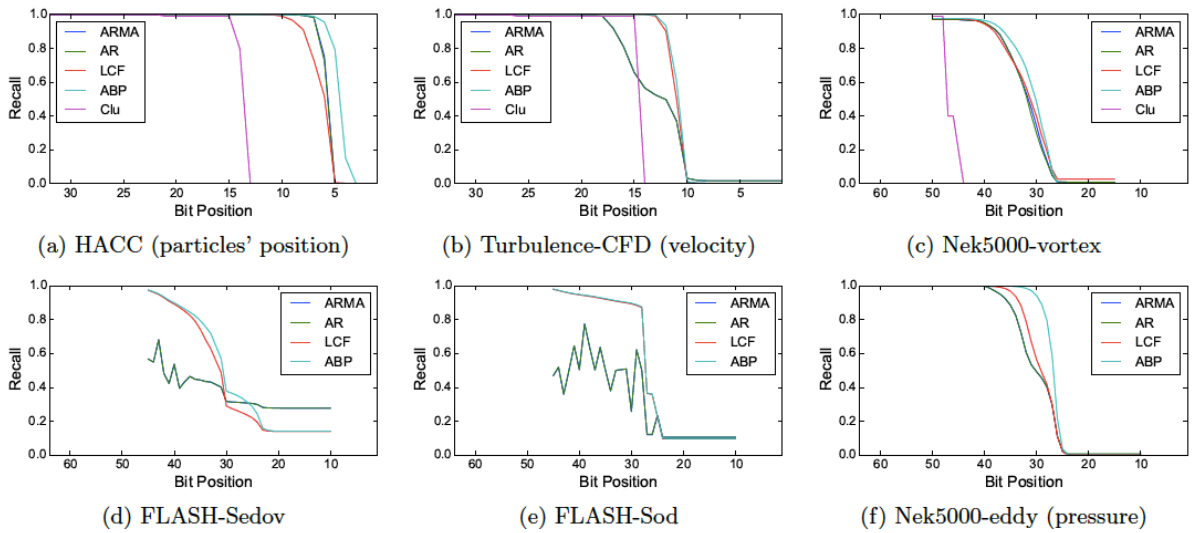



Fig. 2: Ability of variable-tracking algorithms to detect corruptions in bit positions for 4 application codes

For the best detection algorithms, the recall (percentage of detected corruptions compared with all corruptions) is close to 100% up to bit position 5 in HACC and 10 in turbulence CFD; both use a 32-bit floating-point number representation. Recall is >60% for bit position 30 for Nek5000 and FLASH. For all these applications, the detection covers corruptions at the expected user accuracy: $10^{-6}$ for HACC and $10^{-8}$ for Nek5000.

We believe that on-line data analytics of simulation variables opens promising low-cost opportunities for corruption detection in very large-scale numerical simulations and that more research is needed to design and develop detection algorithms for applications where variables have less smooth behaviors.

[1] J. P. J. Kelly, T. I. McVittie, and W. I. Yamamoto, "Implementing design diversity to achieve fault tolerance," IEEE Software, pp. 61–71, 1991.

[2] B. Randell, “System structure for software fault tolerance,” IEEE Trans. Software Eng., vol. 1, no. 2, pp. 220–232, June 1975.
[3] S. Di, E. Berrocal, and F. Cappello, “An efficient silent data corruption detection method with error- feedback control and even sampling for HPC applications,” IEEE CCGRID 2015.