

Adaptive Impact-Driven Detection of Silent Data Corruption for HPC Applications

Sheng Di, *Member, IEEE*, Franck Cappello, *Member, IEEE*

Abstract—For exascale HPC applications, silent data corruption (SDC) is one of the most dangerous problems because there is no indication that there are errors during the execution. We propose an adaptive impact-driven method that can detect SDCs dynamically. The key contributions are threefold. (1) We carefully characterize 18 HPC applications/benchmarks and discuss the runtime data features, as well as the impact of the SDCs on their execution results. (2) We propose an impact-driven detection model that does not blindly improve the prediction accuracy, but instead detects only influential SDCs to guarantee user-acceptable execution results. (3) Our solution can adapt to dynamic prediction errors based on local runtime data and can automatically tune detection ranges for guaranteeing low false alarms. Experiments show that our detector can detect 80-99.99% of SDCs with a false alarm rate less than 1% of iterations for most cases. The memory cost and detection overhead are reduced to 15% and 6.3%, respectively, for a large majority of applications.

Index Terms—Fault Tolerance, Silent Data Corruption, Exascale HPC



1 INTRODUCTION

Researchers are increasingly relying on massively parallel supercomputing to resolve complex problems.

In exascale HPC executions, unintended errors are inevitable because of the huge size of the resources (such as CPU cores and memory). Compared with the fail-stop errors (like hardware crashes), silent data corruption (SDC) is hazardous as there is no indication that the data are incorrect during the execution. A typical example is bit-flip errors striking the memory because of unexpected or uncontrolled factors such as alpha particles from package decay or cosmic rays. Other errors (such as in floating point operators [1]) may not be detected by hardware because of the cost of protective techniques, leading to incorrect compute results at the end of the execution. Accordingly, timely effective detection of the SDC is crucial for guaranteeing the correctness of execution results and high performance.

In our previous work [2], [3], [4], we proposed an SDC detector that predicts the next-step value for each data point and compares the corresponding observed value with a *normal value range* (a.k.a., *detection range*) based on the predicted value, for detecting possible data anomalies. Like most of the existing research [5], [6], [7], we endeavored to optimize both the detection precision (the fraction of true SDCs detected over all detected ones) and sensitivity (the fraction of true SDCs detected over all SDCs experienced). The sensitivity is also known as recall, and we will interchangeably use them in the following text. We compared different linear-prediction methods with regard to HPC runtime data in [3], and we further proposed an error-feedback control model to improve the detection ability in [8].

In this work, we revisit the SDC detection issue and propose an impact-driven model based on our careful char-

acterization of 18 HPC applications/benchmarks. We argue that one should not blindly enhance the detection ability with regard to the SDC. Instead, our research objective is to keep fairly low false alarms (false positives) with acceptable compute results. That is, some SDCs are acceptable, as long as their impact is low enough from the perspective of users. On the one hand, there is a tradeoff between the detection sensitivity and detection precision, since one cannot avoid the data prediction errors. On the other hand, pursuing high detection sensitivity inevitably induces huge detection overhead, as discussed in [3], [8].

At least two challenges arise in this research.

- *Irregularity of SDCs*: Predicting SDCs in practice is difficult because of the random occurrence of SDCs. Thus, it is non-trivial to quantify the impact of the SDCs on the final execution results. Without an in-depth analysis of the impact of SDCs, it is hard to determine an appropriate detection range for detecting the SDCs. The existing research, such as [3], optimizes the detection range based on the desired final compute accuracy. Such a method has a fairly high detection sensitivity, however, it may suffer high false alarms (i.e., low precisions) because the required accuracy may change with iterations or input parameters.
- *Diverse data features of HPC applications*: Since HPC applications are of many different types from different communities (such as physics, chemistry, biology, and mathematics), the iterative runtime data generated during the execution is extremely hard to uniformly model or regularize. Consequently, the features of HPC data would be fairly diverse with applications. For instance, the values of data points may change sharply in some iterations. The overall data value range may also change over time steps. Hence, the data prediction methods may have largely different prediction accuracies with various applications. Clearly needed

• Sheng Di and Franck Cappello are with the Mathematics and Computer Science (MCS) division at Argonne National Laboratory, USA.

TABLE 1: Applications/benchmarks Used in the Characterization

Domain	Name	Code	Description
HD	Blast2 [18]	Flash	Strong shocks and narrow features
	SodShock [19]	Flash	Sodshock tube for testing compressible code's ability with shocks & contact discontinuities
	Sedov [20]	Flash	Hydrodynamical test code involving strong shocks and non-planar symmetry
	DMReflection [18]	Flash	Double Mach reflection: an evolution of an unsteady planar shock on an oblique surface
	IsentropicVortex [21]	Flash	2D isentropic vortex problem: a benchmark of comparing numerical methods for fluid dynamics
	RHD_Sod [22]	Flash	Relativistic Sod Shock-tube: involving the decay of 2-fluids into 3-elementary wave structures
	RHD_Riemann2D [23]	Flash	Relativistic 2D Riemann: exploring interactions of four basic waves consisting of shocks, etc.
	Eddy [24]	Nek5k	2D solution to Navier-Stokes equations with an additional translational velocity
MHD	Vortex [25]	Nek5k	Inviscid Vortex Propagation: tests the problem in earlier studies of finite volume methods [26]
	BrioWu [27]	Flash	Coplanar magneto-hydrodynamic counterpart of hydrodynamic Sod problem
	OrszagTang [28]	Flash	Simple 2D problem that has become a classic test for MHD codes
	BlastBS [29]	Flash	3D version of the MHD spherical blast wave problem
	GALLEX [30]	Nek5k	Simulation of gallium experiment (a radiochemical neutrino detection experiment)
BURN	Cellular [31]	Flash	Burn simulation: cellular nuclear burning problem
GRAV	DustCollapse [32]	Flash	Selfgravitating problems in which the flow geometry is spherical without gas pressure
	MacLaurin [16]	Flash	MacLaurin spheroid (gravitational potential at the surface/inside a spheroid)
DIFF	ConductionDelta [16]	Flash	Delta-function heat conduction problem: examining the effects of Viscosity
	HeatDistribution [33]	customized	Steady-state heat distribution with Laplace's equation by Jacobi iterative method

is an adaptive solution that is suitable for different applications. Moreover, the application codes are made by different programming languages based on various data formats, which also increases the difficulty of our data analysis.

In this paper, we propose a novel SDC detection solution based on the comprehensive characterization of 18 HPC applications/benchmarks on a real cluster - namely FUSION [36]. The key contributions are summarized as follows:

- We carefully study the key HPC data features of the 18 applications, and we characterize the impact of SDC on HPC execution results based on dynamic value ranges and data fluctuation over time.
- We propose a relatively generic impact-driven detection model to detect influential SDCs for the iterative time-step based scientific simulations. The key feature is that it can reduce false positives significantly, and the detection sensitivity is also tunable for different environments with various SDC rates.
- We devise an adaptive SDC detection approach, under which each process can adaptively select the best-fit prediction method based on its local runtime data. Such a solution creates an avenue to adapt to the data dynamics, optimizing the tradeoff between the detection overhead and false alarms.
- We carefully implement the adaptive impact-driven SDC detection library, supporting a broad range of HPC applications coded in either C or Fortran. The library is available to download from [40] under the BSD license. Our detector can detect any SDCs, including not only bitflip errors, but bugs and attacks.
- We evaluate the detector by running HPC applications on up to 1,024 cores. Experiments show that for a large majority of applications, our detection method can keep the rate of false alarms less than 1% of iterations and the detection sensitivity within 80-99.99%, with the memory cost and detection overhead reduced to 15% and 6.3%, respectively.

The rest of the paper is organized as follows. We present the design overview in Section 2. In Section 3, we present our characterization results based on 18 HPC applications across from different codes, and we analyze the key features, such as the smoothness of the time series data and the impact of SDC on HPC execution results. In Section 4, we propose an adaptive solution that can effectively control the

SDC detection overhead and false alarms, while guaranteeing the correctness of the compute results. We present the evaluation results in Section 5. We discuss related work in Section 6 and we present in Section 7 concluding remarks and ideas for future work.

2 SYSTEM OVERVIEW

We illustrate the system architecture in Fig. 1. The key module, fault tolerance toolkit, contains two significant parts, SDC detector and failure/error corrector. In this paper, we only focus on the SDC detection, because the correction issue becomes a real problem only if the detection is already solved well. As for the correction of SDC errors, many existing techniques such as checkpoint/restart [11], [12], [13] have been extensively studied for years. As shown in Fig. 1, our detector will mainly communicate with the HPC application data generated iteratively, in contrast with the algorithm based fault tolerance (ABFT) [14], [15] that is implemented in application library. When an SDC event is detected, for example, the checkpoint/restart model will interrupt the application and restart its execution based on recent one or more checkpoint(s). The reported error would likely disappear in the second run if it was truly an SDC error, or be considered a false positive otherwise.

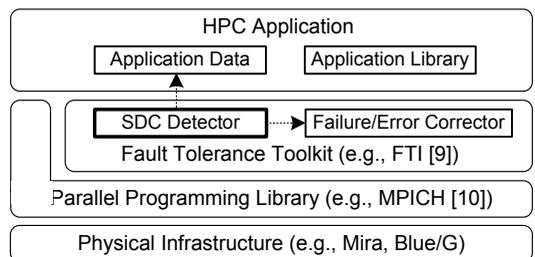


Fig. 1: System Architecture

For the fault model, we focus on the unexpected data changes caused by SDCs such as bit-flips of the data. Our detector performs the one-step ahead prediction only for the *state variables* (a.k.a., *state data points*). The reason is that any time-step based scientific simulations can always restart from a set of variable states (i.e., checkpoint file), based on the characteristics of such type of simulation. That is, the correctness of the state variables' data is a necessary and sufficient condition determining the validity of the remaining execution, no matter where the SDCs happened

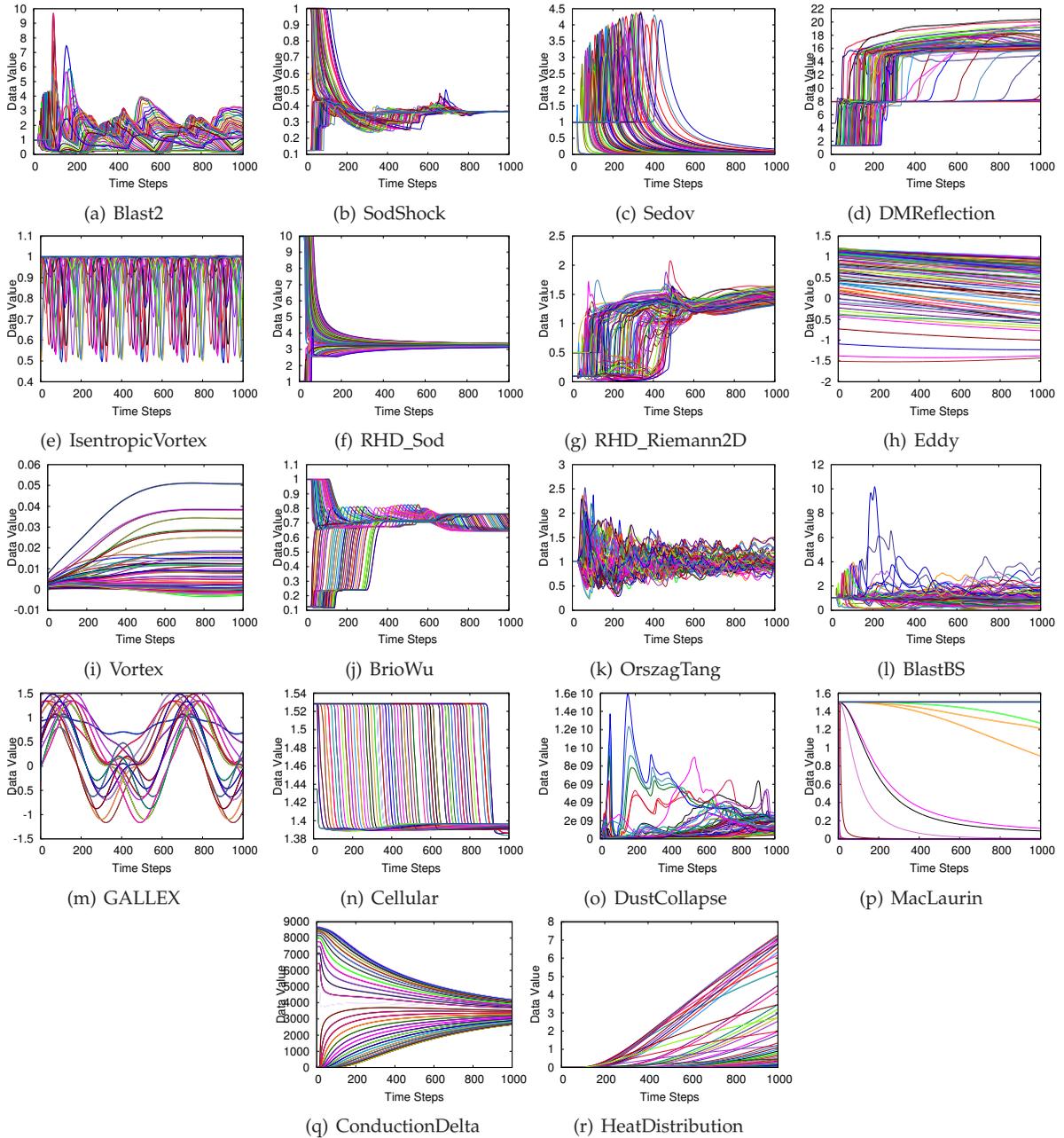


Fig. 2: Sampled Time Series Data of The 18 HPC Applications

in the memory. The users do not need to care about the SDCs occurring outside the state-variable memory (such as intermediary variables and buffers), unless they would affect the state values.

Our research objective is to develop a **generic** SDC detector suitable for a large set of mainstream HPC applications, which perform dynamic simulations over multiple iterations. The basic model used in our detection is one-step ahead prediction [2], [4], [8], which dynamically predicts the value for each data point at each time step and compares the observed value with a normal value range. Unlike previous SDC detectors, our detector aims to detect only **influential** SDCs in terms of dynamic HPC data features. Each process is able to **adaptively** determine the best-fit prediction method based on local runtime data, which can effectively improve the detection sensitivity and reduce overhead.

3 EXPLORATION OF KEY FEATURES FOR HPC APPLICATION DATA

In this section, we characterize the key features of the HPC data regarding SDCs, by running 18 HPC applications/benchmarks on a real cluster environment. In particular, we explore the dynamic HPC data fluctuation and analyze the impact of the SDCs on the execution results respectively, which is the fundamental basis of our adaptive impact-driven detection method.

The 18 real-world HPC applications/benchmarks¹ are

¹Each application here corresponds to a real-world simulation problem. From the perspective of system, each of them is a kind of application that can be submitted by the application user for running in parallel in the system. From the perspective of scientific researchers, maybe it is more appropriate to call some of them benchmarks. So, we will use applications and benchmarks interchangeably in this paper.

from well-known simulation code packages, such as FLASH [34] and Nek5000 (or Nek5k) [35] (except for HeatDistribution that is customized with finite difference methods [33]). All the applications presented in Table 1 involve different research fields, such as hydrodynamics (HD), magneto hydrodynamics (MHD), burning (BURN), gravity (GRAV), and diffusion (DIFF). They are designed using hybrid/semi-implicit methods (except for HeatDistribution that adopts explicit methods) and are coded in either Fortran or C. According to the developers, most of the applications are used to solve real research problems. In this characterization, all of them were run on 128 cores from Argonne FUSION cluster [36]. The number of iterations is set to 1,000, which is large enough to observe the data evolution based on our analysis of execution results.

3.1 Diverse HPC Data Fluctuation

In Fig. 2, we present the time series data fluctuation for each of the 18 HPC applications. Most of the data used in this characterization involve the *density* variable, unless it is unavailable. For instance, neither Eddy nor Vortex has a *density* variable, so we adopt the *pressure* variable instead. The time series data regarding other variables are not presented because of the similar data fluctuation we observed. In the characterization, we select 100 sample points evenly in the data space for each application, and we plot the time series curves for each point by different colors.

Based on Fig. 2, we have four important findings, which indicate high diversity of HPC data with applications.

- 1) *HPC data exhibit different degrees of smoothness.* Some applications exhibit relatively large data changes in short periods (i.e., within a few time steps). In this situation, the data prediction used to detect anomalies may suffer from large errors. By comparison, the data outputted by Vortex are very smooth in the whole period. In this situation, the prediction method will work well, as confirmed by our previous work [8].
- 2) *HPC data have largely different value ranges with applications.* The value range can be split into four categories: tiny range (e.g., [-0.01,0.05] for Vortex), small range (e.g., [0.1,1] for BrioWu), medium range (e.g., [0,20] for majority applications like Blast2, SodShock, and Sedov), and big range (e.g., [0,1.6×10¹⁰] for DustCollapse). Various global value ranges indicate different impacts on the compute results with the same SDC data changes.
- 3) *HPC data usually keep comparative value range in short periods.* For most of the applications (such as Blast2, DM-Reflection, IsentropicVortex, Eddy and OrszagTang), the value range changes slightly, especially in short periods. By contrast, the data value ranges of only a few applications change largely over time, such as Sedov.
- 4) *HPC data exhibit largely different patterns with applications.* Some time series data behave very irregularly, while other application data exhibit a clear periodicity. A typical example with seasonal time series data is IsentropicVortex, as it simulates a periodic phenomenon. Other applications exhibit non-periodic time series data (e.g., ConductionDelta, HeatDistribution, and DustCollapse).

For the detection based on linear prediction methods, the smoothness of the time series data generated by HPC

applications (i.e., the first finding in the above list) is the most significant, because it is closely related to the prediction accuracy. Hence, we study the smoothness of the time series data based on various HPC applications in the following text. We perform the evaluation by both the lag-1 auto-correlation coefficient of the time series (denoted by α) and the lag-1 auto-correlation coefficient of the data changes (denoted by β), since neither can individually represent the smoothness of HPC data, as shown later. We call the two coefficients *level-1 smoothness coefficient* and *level-2 smoothness coefficient* respectively.

The auto-correlation coefficient (ACC) [37] is a well-known indicator to evaluate the auto-correlation of time series data. Its definition is shown in Equation (1), where $V(t)$ refers to the data value at time step t , $E[\cdot]$ and μ refer to the mathematical expectations that can be estimated by mean values, and σ refers to the standard deviation. The value range of the auto-correlation coefficient is $[-1, 1]$, with 1 indicating perfect correlation, 0 indicating noncorrelation, and -1 indicating perfect anticorrelation.

$$\alpha = \frac{E[(V(t) - \mu(V))(V(t+1) - \mu(V))]}{\sigma^2(V)} \quad (1)$$

In addition to the ACC of time series data, we also evaluate the smoothness by the ACC of data changes (i.e., level-2 smoothness), which is defined in Equation (2), where $\Delta(t) = V(t) - V(t-1)$, and $\mu(\Delta)$ and $\sigma(\Delta)$ refer to the expected data change and the deviation of data change, respectively.

$$\beta = \frac{E[(\Delta(t) - \mu(\Delta))(\Delta(t+1) - \mu(\Delta))]}{\sigma^2(\Delta)} \quad (2)$$

In Fig. 3, we give an example to further illustrate the level-2 smoothness coefficient. It is easy to see that $\Delta(i)$ and $\Delta(i+1)$ are equal to the left derivative and right derivative of the data point value at the time step i . The curve is considered smooth at the time point i when its left derivative and right derivative have close values (i.e., the difference $|\Delta(i) - \Delta(i-1)| < \epsilon$). In particular, if the two derivatives are always the same (i.e., $\Delta(i) = \Delta(i-1)$) for each time point, the curve is a straight line.

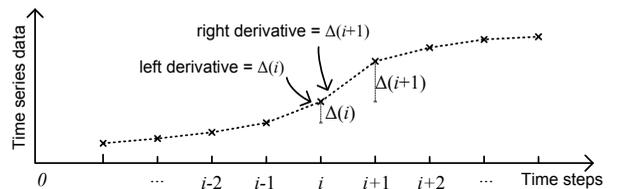


Fig. 3: Smoothness Evaluation by Lag-1 ACC of Data Changes (Level-2 Smoothness Coefficient)

By combining the two levels of smoothness coefficient, we can study the smoothness feature for the HPC time series data comprehensively. The level-1 coefficient α represents the overall smoothness of the time series, while the level-2 coefficient β indicates the existence of the sharp changes in the time series. Specifically, if the time series data exhibit perfectly smooth changes (such as a straight line or a quadratic curve), both coefficients are supposed to be very high (close to 1). If α is close to 1 while β is relatively low, the time series will look smooth overall, with only a few sharp changes at particular time steps.

In Fig. 4, we present the cumulative distribution function (CDF) of the two coefficients for 8 typical applications. For the other 10 HPC applications listed in Table 1, both of the two smoothness coefficients are always greater than 0.95, which means their time series data are always very smooth.

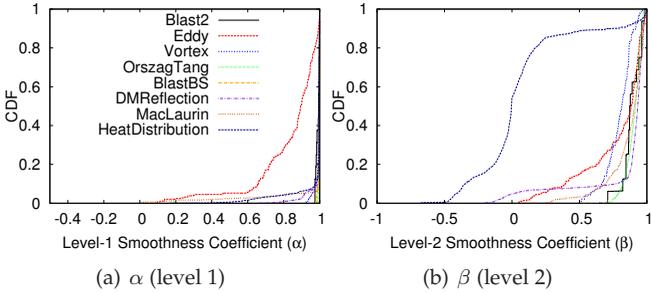


Fig. 4: Distribution of Smoothness Coefficient

In Fig. 4, we can observe that all of the time series data (except Eddy) exhibit smooth changes overall in the whole period. Compared with the level-1 smoothness coefficient, we find that the level-2 smoothness coefficients do not always show very high values for many applications. This means that in the HPC time series data of such applications, sharp changes exist at some time steps, although the data are smooth overall in the whole period. We note that some application data such as IsentropicVortex look not smooth in the whole period in Fig. 2, yet they are not presented in Fig. 4. The reason is that their two coefficient values are always close to 1, indicating that their data actually are very smooth in short periods.

In summary, the HPC time series data of the analyzed applications are smooth overall but may have sharp changes at a few time steps. This finding indicates that the prediction method should work effectively for detecting SDCs at runtime. However, because of the data dynamics as shown above, it is necessary to devise an adaptive method which can suit different fluctuation features of the HPC data.

3.2 Impact of the SDCs on Execution Results

In this subsection, we first analyze the factors of the impact of SDC on HPC executions, based on which we formulate the impact of SDC in Section 3.2.3. We then characterize SDC's impact based on real-world applications, which is the basis of our impact-driven detection method.

3.2.1 Factors Affecting SDC Impact

Basically, three factors directly determine the impact of the SDC on HPC execution. We will use them to formulate the impact of SDC later.

- 1) *Bit position of the data value flipped.* A single-precision floating-point number is represented by 32 bits in binary and a double-precision number by 64 bits in binary. Different bit positions flipped by the SDC will affect the data differently. Taking the number 1.0 as an example, upon the bit-flip error with the bits 10, 20, 30, 40, and 50, the new values are $1+2.274 \times 10^{-13}$, $1+2.328306 \times 10^{-10}$, 1.00000023841858, 1.000244140625, and 1.25, respectively.
- 2) *Value range of the state variable.* For the same data change induced by SDC, various compute data value

ranges also affect the impact of the SDC differently on the execution results. For example, suppose the data change is 0.5 due to a bit-flip error. Its impact would be easily observed if all data values appear in a small range such as $[0,1]$, whereas it would be hardly perceived in a fairly large value range such as $[0,10000]$.

- 3) *Occurrence moment of the SDC during the execution.* The impact of SDC on the final execution result will be different when the SDC occurs at different moments (such as at the beginning versus at the end of the running period). The reason is that the SDC would lead to different impacts on the runtime outputs at different following time steps after the corruption, as shown later on (in Section 3.2.4).

3.2.2 Preliminary Observation of SDC Impact

Fig. 5 illustrates SDC impact on runtime outputs by running DMReflection with injected bit-flip errors. As shown, each snapshot is split into $8 \times 16 = 128$ tiles, which were computed on 128 cores. One error was injected at step 20 to the bottom middle point $(2,0,0)$ with bit 56 flipped, such that the value was changed from 1.7856 to 2.7428×10^{-5} . Through the figure, we observe the impact of the SDC on the execution result would be submerged with time going, i.e., the maximum difference between the fault-free execution and SDC-based execution decreases over time. This is because of the mutual influence of the bit-flipped data point and its surrounding data points, such that the impact of the bitflip error on the execution result would be mitigated in the following computation over time. Such an observation confirms that the occurrence moment of SDC during the execution is one key factor affecting the impact of SDC.

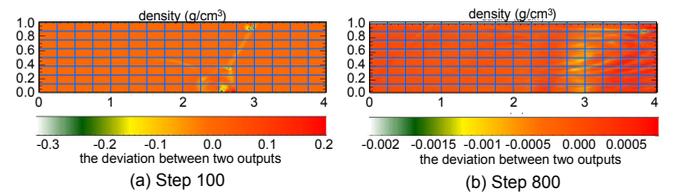


Fig. 5: Impact of SDC on DMReflection

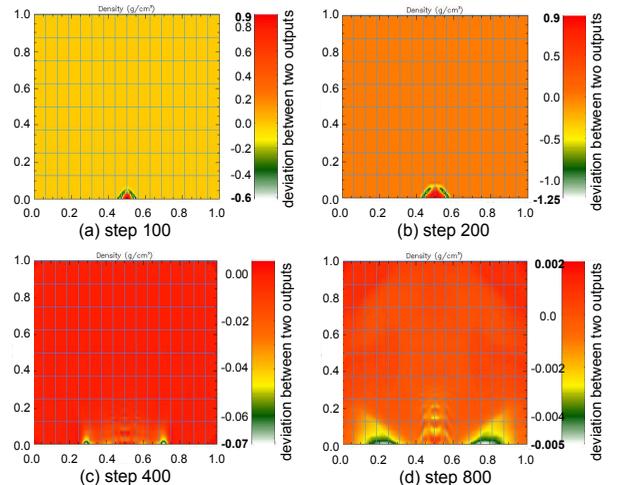


Fig. 6: Impact of SDC on Sedov (a bottom point of value 2.5×10^{-5} has an error on bit 56 at step 20)

In fact, the deviation of the runtime outputs compared to the fault-free outputs may not always decrease after the corruption. Fig. 6, for example, presents the impact of a bit-

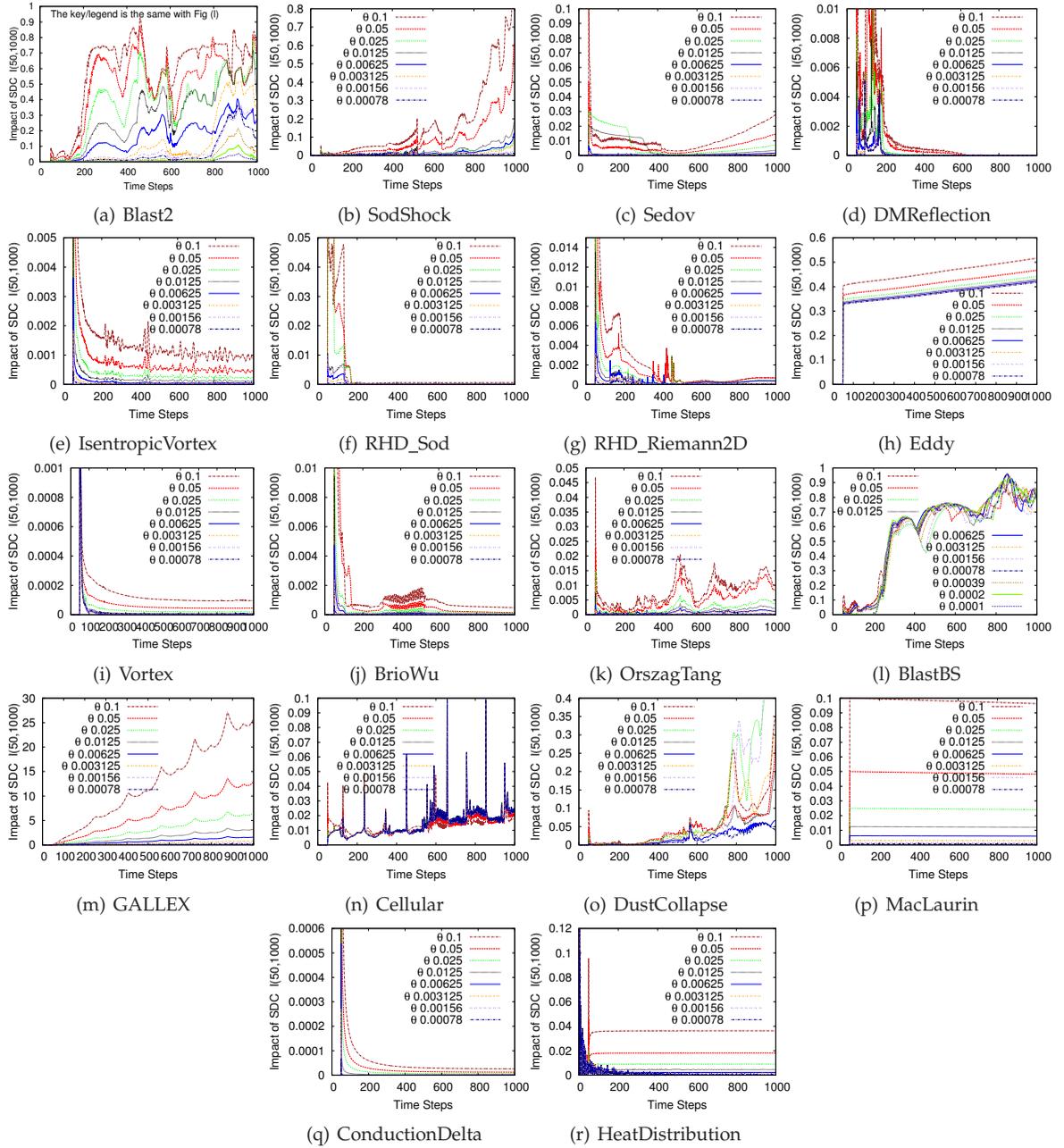


Fig. 7: Impact of the SDC on HPC Applications with Different Injected Bit-flip Errors

flip error on Sedov's execution. The bit-flip error was also injected at time step 20, and it flipped the bit position 56 of the bottom middle data point, such that its value was changed from 2.5×10^{-5} to 1.6384 silently. We can observe that the value range of the deviation between the original fault-free output and the bit-flip induced output is about $[-1.25, 0.9]$ at step 200, in the comparison of the range $[-0.6, 0.9]$ at step 100. More discussion of corruption propagation can be found in our previous work [3].

3.2.3 Formulation of SDC Impact

Based on the three key factors summarized previously, we formulate the impact of SDC on execution results (denoted by I) as the maximum ratio of the absolute data change value to the overall value range during a period after the corruption. Suppose the SDC occurs at time step t_1 . The impact of the SDC during the period t_1 through t_2 is defined

in Equation (3), where δ_t refers to the absolute deviation between the SDC-free output and SDC-induced output at time step t , and r_t refers to the value range size at time step t (i.e., $r_t = \max(V_t) - \min(V_t)$).

$$I(t_1, t_2) = \max_{t \in [t_1, t_2]} \left\{ \frac{\delta_t}{r_t} \right\} \quad (3)$$

3.2.4 Characterization of SDC Impact

We characterize the impact of SDCs on the 18 HPC applications listed in Table 1, based on the above definition of SDC impact. Each application was run by 128 cores for 1,000 iterations (i.e., time steps), and one bitflip error was injected at time step 50 onto randomly selected data points with various degrees of data changes between adjacent time steps. We traverse 8 testcases with different value changes caused by the SDC for each application. The SDC-induced value change is evaluated by the *relative data change ratio* (denoted by ϑ_{sdc}), which is defined as the ratio of some data

point's absolute value change induced by SDC (denoted by Δ_{sdc}) to the global value range (denoted r_{sdc}) at the injection moment. In our characterization, the value of ϑ_{sdc} decreases exponentially ($=0.1 \times 2^{-m}$, where $m=0,1,2,\dots$) for observing the impacts of SDCs with exponential value changes on different data points. Specifically, $\vartheta_{sdc}(=\frac{\Delta_{sdc}}{r_{sdc}})$ is set to 0.1, 0.05, 0.025, 0.0125, 0.00625, 0.003125, 0.0015625 and 0.00078125 in the 8 testcases, respectively. We run each case with the same level of value changes 10 times, by randomly selecting the data points at different locations of the snapshot.

In Fig. 7, we present the impact of SDCs in the whole remaining period (i.e., $I(50,1000)$). Two types of impact traces exist. (1) The impacts of the SDCs on majority (15 out of 18) of applications are basically proportional to ϑ_{sdc} . (2) Only three applications (Eddy, BlastBS, and DustCollapse) are very sensitive to tiny SDC-induced data change, which is due to very close mutual relations among neighboring data points in both space and time.

3.2.5 Controlling the Impact of SDCs

To control the impact of the SDCs on demand, we introduce an *impact error bound ratio* θ , which is defined as the bound of the relative data change ratio that makes sure the impact of SDCs can be limited to a low level in the whole execution period. The impact error bound ratio is formally defined in Equation (4), where $[t_{sdc}, t_{end}]$ refers to the whole execution period after the corruption and φ indicates the acceptable maximum impact of the SDC in the whole execution period. For instance, based on Fig. 7 (b), (c), and (d), if the relative data change ratio induced by SDC is always below 0.00078125 for SodShock, Sedov and DoubleMachReflection, respectively, the impact of the SDC would be strictly limited within 2%, 0.09%, and 0.5% respectively, in the whole execution period. That is, such an impact error bound ratio can be leveraged to design an impact-driven detector for detecting only influential SDCs in the execution (discussed in details in Section 4).

$$\theta = \max_{I(t_{sdc}, t_{end}) \leq \varphi} \{\vartheta_{sdc}\} = \max_{I(t_{sdc}, t_{end}) \leq \varphi} \left\{ \frac{\Delta_{sdc}}{r_{sdc}} \right\} \quad (4)$$

Based on Fig. 7, we find that a large majority of applications (15 out of 18) are suitable to be protected based on the impact error bound ratio. When the impact error bound ratio θ is set to 0.00078125, the impact of SDC can be limited below 2% of the data value range in the whole execution period for most cases (an exception is Blast2, whose bound ratio recommended is 0.0001). Thus, $\theta=0.00078125$ or 0.0001 is a recommended bound ratio when the users have no preliminary impact traces for their applications.

4 ADAPTIVE IMPACT-DRIVEN SDC DETECTOR FOR HPC APPLICATIONS

The basic idea is to allow different running processes to select the best-fit prediction methods for their own detections, in terms of their local runtime data. Fig. 8 presents the compute result of the Sedov shockwave simulation at time steps 100 and 200. As illustrated, the whole data set is split into $16 \times 8 = 128$ tiles, which were processed by 128 processes in parallel. One can clearly see that the data

handled by different processes evolve differently at different time steps. In particular, the data around the edge of the shockwave change sharply, introducing greater difficulty for data prediction than other data areas require. Therefore, the processes are supposed to adopt different prediction methods with various accuracies and overheads. For instance, at time step 100, rank 14, rank 25, and rank 36 adopt last state fitting (LSF), linear curve fitting (LCF), and quadratic curve fitting (QCF), respectively. At time step 200, the prediction methods of rank 25 and rank 36 are changed because of the changed data fluctuation around that moment. In our design, the prediction methods are automatically changed based on the recent prediction errors estimated dynamically, to be discussed in details in Section 4.3.1.

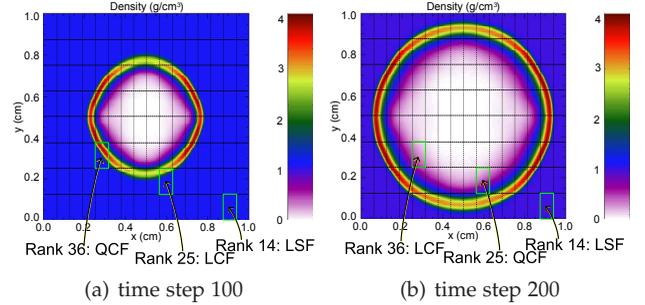


Fig. 8: Illustration of Data Partitioning and Adaptive Prediction (Sedov)

4.1 Detection Model

The overall detection model is illustrated in Fig. 9. Our detector checks each local data point at each time step. For any specific data point, we first perform the one-step ahead prediction based on the bestfit prediction method, and construct a normal value range in terms of the impact error bound explored based on the characterization of real-world applications (see Fig. 7). The normal value range is set to $[X(t) - \rho, X(t) + \rho]$, where $X(t)$ is the predicted data point value for the time step t and ρ is called *detection radius*. Then, we compare the observed data value $V(t)$ (the circle point in the figure) with the normal value range for detecting the possible SDCs. Two types of predictions exist, valid prediction (the prediction error is smaller than the impact error bound) and invalid prediction (prediction error is greater than the impact error bound), as shown in Fig. 9. Our detector seeks to select the prediction method with high prediction accuracy and low memory cost if valid prediction methods exist, or otherwise it will be gracefully degraded to choose the simplest prediction method with lowest memory cost. The detection range can also be dynamically enlarged upon the gracefully degraded situation with large prediction errors. Such a design can effectively reduce the false positives and control the memory overhead.

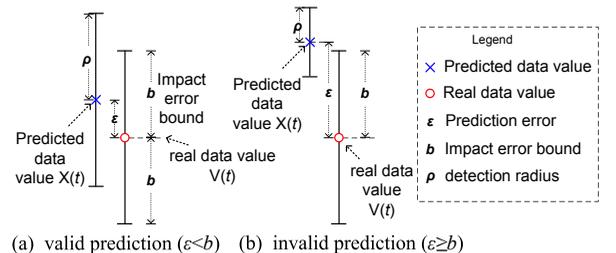


Fig. 9: Detection with Valid/Invalid Prediction

4.2 Error Feedback Prediction

In our detector, we use curve fitting to perform one-step ahead prediction in the detector. Since it is non-trivial to derive high-order prediction formulas for curve fitting, we explore a **recursive error feedback** model that can significantly simplify the prediction. We also demonstrate the identity between the feedback prediction and curve fitting prediction, in comparison to our previous work [8].

We denote $X(t)$ the data value to be predicted at the time step t , and the detector keeps N recent observed values (denoted by $V_{t-1}, V_{t-2}, \dots, V_{t-N}$) for the data point. Then, the error feedback prediction formula (with the feedback order being equal to N) can be presented in recursive form, as shown in Equation (5), where $a_{t-i}^{(N)}$ denotes the coefficient of the N th-order feedback prediction at time step $t-i$, and $e_{t-1}^{(N-1)}$ is the prediction error at time step $t-1$. The initial state, the 0th-order feedback prediction $X^{(0)}(t)$, is set to $V(t-1)$ (i.e., last state fitting (LSF)).

$$\begin{aligned} X^{(N)}(t) &= \sum_{i=1}^N \left(a_{t-i}^{(N)} V(t-i) \right) = \sum_{i=1}^{N-1} \left(a_{t-i}^{(N-1)} V(t-i) \right) - e_{t-1}^{(N-1)} \\ &= \sum_{i=1}^{N-1} \left(a_{t-i}^{(N-1)} V(t-i) \right) - \left(\sum_{i=1}^{N-1} a_{t-1-i}^{(N-1)} V(t-1-i) - V(t-1) \right) \end{aligned} \quad (5)$$

Some examples of low-order feedback predictions are listed below, based on the above Equation (5).

$$\begin{aligned} X^{(1)}(t) &= V(t-1) - [V(t-2) - V(t-1)] = 2V(t-1) - V(t-2) \\ X^{(2)}(t) &= 3V(t-1) - 3V(t-2) + V(t-3) \\ X^{(3)}(t) &= 4V(t-1) - 6V(t-2) + 4V(t-3) - V(t-4) \\ X^{(4)}(t) &= 5V(t-1) - 10V(t-2) + 10V(t-3) - 5V(t-4) + V(t-5) \end{aligned}$$

In the following proposition, we prove the identity between the feedback predictions and the curve fitting methods up to the third order, since higher orders do not help reduce prediction errors as we observed in experiments.

Proposition 1. (1) First-order feedback prediction is identical to linear curve fitting (LCF). (2) Second-order feedback prediction is identical to quadratic curve fitting (QCF). (3) Third-order feedback prediction is identical to cubic curve fitting (CCF).

Proof: (1) The LCF plots a linear line (denoted by $f(x) = ax + b$) based on the recent two observed values for the target data point to make the prediction, as shown in Fig. 10 (a). Because of the equal length of the time steps, the coordinates of the two recent data values can be denoted as $(0, V(t-2))$ and $(1, V(t-1))$, respectively. Then the predicted value at t is equal to $f(2) = 2a + b$, where a and b can be computed by $(0, V(t-1))$ and $(1, V(t-2))$, respectively. One can easily verify that $f(2) = 2V(t-1) - V(t-2) = X^{(1)}(t)$.

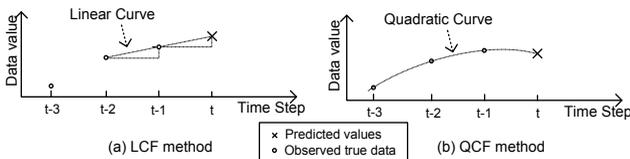


Fig. 10: Illustration of LCF and QCF

(2) For QCF, a quadratic curve (denoted by $f(x) = ax^2 + bx + c$) will be determined based on the recent three values, which can be denoted as $(0, V(t-3))$, $(1, V(t-2))$, and $(2, V(t-1))$, respectively, as shown in Fig. 10 (b). Then, the predicted value at t can be computed by $f(3) = 9a + 3b + c$, where

a, b , and c are computed by the recent three values. One can easily verify $f(3) = 3V(t-1) - 3V(t-2) + V(t-3) = X^{(2)}(t)$.

(3) We also validated the identity of CCF and the 3rd-order feedback prediction. That is, the predicted value in terms of the cubic curve plotted by the recent four values is right equal to the 3rd-order feedback prediction value $X^{(3)}(t) = 4V(t-1) - 6V(t-2) + 4V(t-3) - V(t-4)$. \square

Remark: (1) Based on Proposition 1, we can draw a reasonable conjecture that the N th-order feedback prediction is identical to the N th-order curve fitting. The strict proof will be studied in our future work. (2) The feedback prediction methods with various orders will lead to different prediction accuracies/errors. In principle, higher-order predictions may lead to higher prediction accuracy, yet this may not be true for some applications based on our characterization. As shown in Table 2, 13 applications (highlighted in bold) out of the 18 applications have higher prediction accuracy when using the 2nd order feedback prediction (i.e., QCF) than using the 1st order feedback prediction (i.e., LCF) or 0th order feedback prediction. However, the prediction error may not always decrease with the prediction orders for some applications, such as HeatDistribution. Thus, we have to design an adaptive solution based on the runtime prediction errors, as detailed later.

4.3 Adaptive Impact-Driven Detector (AID)

4.3.1 Adaptive Selection of Best-fit Prediction Method

Initially, we wanted to leverage the data changes to anticipate the prediction errors, in that the prediction errors might be consistent with the data changes in general (i.e., larger data changes may lead to larger prediction errors). However, this idea is not feasible, since our characterization shows that the data change vs. prediction errors are not consistent in many cases. Specifically, the Pearson product-moment correlation coefficient (PPMCC)¹ of the data change vs. prediction errors is only 0.5 for majority of applications, and it is even smaller than 0 for a few applications (such as ICE model). Hence, we have to explore a more effective approach to select the best-fit prediction methods at runtime.

Our solution involves two phases for each particular process to search the best-fit prediction method: (1) filtering out the invalid prediction methods based on the impact error bound and local runtime data and (2) selecting the best-fit prediction method based on memory cost and prediction errors. We adopt the feedback prediction methods with different orders as the candidate prediction methods.

The pseudo-code for searching the bestfit prediction method is presented in Algorithm 1. We first aggregate the global data value range (denoted by r) for each process, by leveraging the collective MPI function `MPI_Allreduce` (line 1). We show in the next paragraph that the communication cost is negligible for all applications considered in this study and also could be controlled by users. Then, the current rank/process will estimate its maximum local prediction error (denoted by ε_j) for each candidate prediction method (denoted by $PM_j, j=1,2,\dots,n$), based on the local data set S . At line 5, the solutions whose maximum local prediction errors estimated are smaller than the acceptable impact error

¹PPMCC is a well-known measure of the linear correlation between two variables, giving a value in the range $[-1,1]$.

TABLE 2: Mean Linear Prediction Errors

App.	0th order	1st order	2nd order	App.	0th order	1st order	2nd order
Blast2	0.017	0.006	0.005	BrioWu	7.6×10^{-4}	1.3×10^{-4}	8×10^{-5}
SodShock	0.001	2.9×10^{-4}	2.7×10^{-4}	OrszagTang	0.0074	0.0019	0.0013
Sedov	0.006	0.0008	0.0004	BlastBS	0.003	7.6×10^{-4}	5.5×10^{-4}
DMRefle.	0.017	0.0096	0.0108	GALLEX	0.0064	9.6×10^{-5}	2×10^{-6}
Isen.Vortex	0.0014	1.7×10^{-4}	5×10^{-5}	Cellular	2.2×10^5	1.1×10^5	1.4×10^5
RHD_Sod	0.006	0.0027	0.003	DustColl.	8×10^6	1×10^6	6×10^5
RHD_Rie.	0.0024	0.0004	0.0002	MacLaurin	1×10^{-7}	7×10^{-11}	1.2×10^{-10}
Eddy	7×10^{-5}	2.9×10^{-8}	2.7×10^{-11}	Cond.Delta	3.5	0.136	0.08
Vortex	4×10^{-6}	2.8×10^{-8}	1.7×10^{-8}	HeatDistri.	0.012	0.0075	0.0138

bound θr will be selected to construct a valid solution set Γ (also as shown in Fig. 9). If Γ is empty, the algorithm will choose the prediction method with minimum memory cost (line 13-14); otherwise, it will try constructing an outstanding prediction method set Γ' (line 2), in which the prediction errors are strictly limited below $\lambda\theta r$, where $\lambda \in (0,1)$ is an adjustment coefficient. We evaluate the detection effect using various λ (such as 0.1, 0.2, 0.3, ...) in our experiments (discussed later). The method with the minimum memory cost will be selected from Γ' if it is not empty (line 9), or otherwise the method with the minimum prediction error will be selected from Γ (line 11).

Algorithm 1 ADAPTIVE BEST-FIT PREDICTION METHOD

Input: current time step t , the local data set (denoted S) managed by the current rank i , impact error bound ratio (denoted θ) to avoid influential SDC impact, the set of n prediction methods (denoted $\Pi = \{PM_1, PM_2, \dots, PM_n\}$)

Output: bestfit prediction method.

```

1: Compute global data range  $r$ .
2: for ( $PM_j, j=1, 2, \dots, n$ ) do
3:   Compute max local predict error  $\varepsilon_j$  by sample points.
4: end for
5: The valid solution set  $\Gamma = \{PM_j \mid \varepsilon_j < \theta r\}$ .
6: if ( $\Gamma \neq \Phi$ ) then
7:   Construct the outstanding set  $\Gamma' = \{PM_j \mid \varepsilon_j < \lambda\theta r\}$ 
8:   if ( $\Gamma' \neq \Phi$ ) then
9:     Output the method with min memory cost from  $\Gamma'$ .
10:  else
11:    Output the method with min predict error from  $\Gamma$ .
12:  end if
13: else
14:   Output the method with min predict error from  $\Pi$ .
15: end if

```

Remark:

(1) The communication cost of *MPI_Allreduce* could be controlled on demand. According to [38], [39], the cost of *MPI_Allreduce* increases with message size. In our algorithm, only two numbers (max and min) need to be aggregated for each variable and the total number of key variables to protect is usually small (e.g., 10 for FLASH, 4 for Nek5000, and only 1 for HeatDistribution), so the communication cost would be relatively small. On the other hand, since the global data value range does not change largely during a short period (the 3rd finding in Section 3.1), the collective operation can be performed periodically for further reducing the communication overhead. The users can also avoid the communication cost, by either adopting non-blocking collective operations or setting a static value range if the global value range is actually fixed.

(2) The memory cost could be controlled as well when the data changes slightly or sharply. When the majority of data vary slightly over consecutive time steps, only the 0th

order prediction would be adopted for most data points, suffering very limited memory cost. If the data change sharply in short periods such that the prediction errors of all methods are larger than the impact error bound, our Algorithm 1 would be gracefully degraded to use the simplest prediction method with the minimum cost.

(3) The aggregated global value range may be affected by SDC in two ways. For the first way, the data change induced by SDC may be so large that the global maximum/minimum value is affected. We can strictly prove that the detection sensitivity will not be affected clearly. We omit the strict proof, but give a description because of the space limitation. On the one hand, suppose the value range decreases unexpectedly due to SDC, the impact error bound for all data points will be reduced and thus the detection range will be shortened, leading to a higher detection sensitivity. On the other, if the global value range increases significantly due to SDC, the value of the data point affected significantly by SDC can be easily perceived by the detector. As for the second way, the SDC might occur during the calculation of global value range, which can be resolved by computing the global value range twice or more times because of tiny execution overhead (total execution overhead is $\leq 6.3\%$ for most applications, to be shown in Section 5.2).

(4) The probability of the detector itself being struck by SDC is very small (about 1%), because the memory cost is only about 1% of the memory footprint (as shown in Section 5.2). In fact, we can further improve the reliability by replicating the detector, since the probability of having the replicated detectors hit by SDCs the same way at the exactly same time is negligible.

(5) The sample points used to estimate the best-fit prediction method periodically (line 3) are selected based on even-sampling method proposed in our previous work [8].

(6) The candidate prediction methods are the feedback predictions with various orders. If the latest best-fit prediction order is low (e.g., if it is only 1st-order), the high-order methods (such as 2nd-order method) cannot be checked immediately, since they require more time steps of values than do the low-order methods. To this end, our algorithm periodically stores all recent values (four recent time steps) for the sampled points, for keeping the high-order prediction methods always available to check. The periodic length is set to 20 time steps in our experiment, since it already leads to satisfactory detection results.

4.3.2 Adaptive Detection Range upon False Positives

After running Algorithm 1, each local data point is checked by comparing the observed value with a normal value range

$[X(t) - \rho, X(t) + \rho]$, which is constructed by the selected best-fit prediction value $X(t)$ and a detection radius ρ . The time step t is considered having SDC if and only if the observed value of some data point at the time step t falls outside the predicted normal value range.

The detection radius devised in our solution can dynamically change in order to further reduce false positives, which is motivated by the fact that many false positive events may appear consecutively (based on our observation). As the detector detects an SDC in the execution, the application is rolled back to recent checkpoints for recovery, following the method presented in [13]. If the reported SDC appears again in the second run, it will be marked as a false positive/alarm; the corresponding step is called *false positive step/iteration*. Our adaptive detection radius is enlarged upon a false positive event, as presented in Equation (6). In this equation, η refers to the number of false positive iterations encountered in the past, ε is referred to the prediction error and θr is the user-required error bound with regard to the relative bound ratio θ and runtime data value range r . As derived in our previous work [3], $\varepsilon + \theta r$ is the optimal detection radius which may lead to zero false positives. Note that ε refers to the prediction error, which is approximated in our implementation by using the maximum prediction error of the local data points for each rank at the most recent bestfit order computation time-step.

$$\rho = (1 + \eta)(\varepsilon + \theta r) \quad (6)$$

4.4 Implementation

We implement the adaptive impact-driven detector strictly based on our design. It can also be integrated with the FTI library [9], such that the users are allowed not only to detect the SDCs but to correct the errors by checkpoint/restart model. Our implementation provides both C and Fortran interfaces, such that a broad range of HPC applications can work with our detector. The library is available to download from [40]. There are only four simple steps for users to annotate their MPI application codes: (1) initialize the detector by calling `SDC_Init()`; (2) specify the key variables to protect by calling `SDC_Protect(var)`; (3) annotate the execution iterations by inserting `SDC_Snapshot()` into the key loop; and (4) release the memory by calling `SDC_Finalize()` in the end. Our detector will then protect the HPC application against SDCs after the compilation.

5 PERFORMANCE EVALUATION

In this section, we first show experimental setup and then present the evaluation results.

5.1 Experimental Setting

We evaluate our adaptive impact-driven detector by running 18 HPC applications/benchmarks on 128-1,024 cores from Argonne FUSION cluster [36]. For each application, all the state variables are protected in our experiments. Since different data points struck by the SDCs lead to various detection results due to data dynamics, we must check each data point at each time step for each application. We focus on the 15 applications whose execution results can be

guaranteed correct by controlling the impact error bound ratio θ . As for the other three applications (Eddy, BlastBS, and DustCollapse), the impacts of SDCs on their execution results are fairly sensitive to tiny SDC-induced data change (as shown in Fig. 7), such that our current detector cannot work effectively. How to detect SDCs for these applications will be studied in the future work.

For detection sensitivity, we check every data point by injecting the errors with different bit flips in binary such that the value changes of the data point are beyond the impact error bound ratio θ (characterized in Section 3.2.4), which is set to 0.0001 for Blast2, 0.05 for HeatDistribution, and 0.00078125 for other applications, since these settings guarantee that the impact of SDC is below 2%, based on our characterization (see Section 3.2.4). A time step is considered false positive as long as there exists one data point whose observed value (without error injection) falls outside the predicted normal value range at that moment. The coefficient λ that determines the outstanding solution set is set to 0.2. In fact, our experiments show that $\lambda=0.1-0.5$ leads to the same detection results, in that the prediction errors are either much smaller than $0.1 \cdot \theta r$ or greater than $0.5 \cdot \theta r$, such that $\lambda=0.1-0.5$ can filter non-outstanding methods easily. Both θ and λ are tunable in a configuration file on demand. How to automatically determine their values is our future work.

5.2 Experimental Results

Fig. 11 presents the cumulative distribution function (CDF) of the false positive rate for 15 applications, based on the execution of 128 processes/ranks. The *false positive rate (FP-rate)* is used to evaluate the detection precision; it is defined as the number of false positive iterations over the total number of iterations under the evaluation. The lower the FP-rate is, the more precise the detection. As shown in Fig. 11 (a), (c), and (d), the FP-rate of our adaptive impact-driven detector (AID) is significantly lower than that of the QCF method and CCF method (i.e., QCF with 1st-order feedback) proposed in [3], [4] and [8], respectively. The two methods were considered the best solutions in those studies compared with other linear-prediction methods, including auto-regression (AR) and auto-regression moving-average (ARMA). The key reason that our solution has a much lower FP-rate is twofold. First, our solution is driven by the relative error bound ratio θ , which can automatically tune the impact error bound upon the change of data value range. That is, as the global data value range increases over time, our solution can increase the detection range to adapt to the increase of the impact error bound. This approach is in contrast with the QCF/CCF method that adopts a fixed detection radius. Second, our solution is able to dynamically adapt to the false positive events at runtime. As shown in Fig. 11 (a) and (b), our FP-adapted design (Section 4.3.2) is able to reduce the FP-rate down to 10% for all test-cases and below 1% for a large majority of cases. That is, there is only 1 iteration with unnecessary recovery triggered every 100 iterations in the execution for a large majority of applications. As shown in Fig. 11 (c) and (e), although the FP-rate can also be reduced for QCF by enlarging the detection radius ρ from 0.0001 to 0.01, this would cause unacceptable low detection sensitivity, to be shown later.

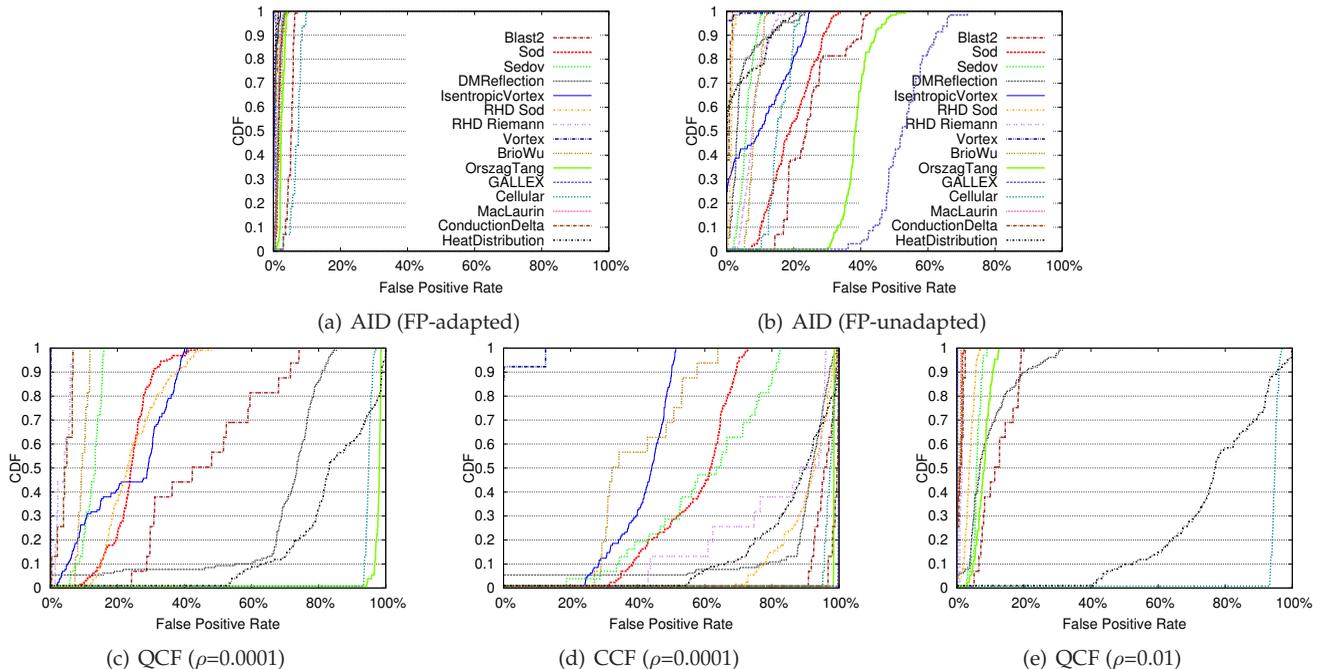


Fig. 11: Distribution of False Positive Rate (Legends of (c),(d),and (e) are the same to (a) and (b))

We present the detection sensitivity of our solution in Fig. 12. Detection sensitivity (i.e., recall) is defined as the fraction of true positives (true alarms) that are detected over all SDCs experienced/injected. In the figure, we can clearly observe that under our FP-adapted AID and FP-unadapted AID, the detection sensitivity is about 80% and 95% for most of the detections, respectively. The detection sensitivity can reach up to 99.99% for a few applications. By comparison, the sensitivities of the QCF detectors with $\rho=0.0001$ and $\rho=0.01$, are around 97% and 75%, respectively. Note that the QCF detector with $\rho=0.0001$ is actually unacceptable from the perspective of false positive (as shown in Fig. 11), so the best solution is our FP-unadapted AID from the perspective of recall. By combining Fig. 11 (a) and (b) and Fig. 12 (a) and (b), users are allowed to select one of the two versions of AID on demand, with various detection preferences (either lower FP-rate or higher recall).

In what follows, we discuss the detection overhead, including memory overhead and execution overhead. We will see that the memory overhead of our detector is $\leq 15\%$ of the amount of memory occupied by the applications at runtime and the execution overhead (including computation cost and communication cost with MPI_Allreduce) is $\leq 13.5\%$ of the execution time without our detector.

In Fig. 13 (a), we present the memory cost of our detector, as compared to the memory size occupied by only state variables. Compared with the QCF whose memory cost is always 4X, our detector can reduce the memory overhead by 37.5-67.5%, which is mainly due to the adaptive selection of the best-fit orders during the execution. Fig. 13 (b) presents the best-fit orders selected by rank 0 in the whole execution period (1,000 iterations) for the 15 applications. One can clearly see that the best-fit orders indeed are dynamically changed over time. Most of the best-fit orders stay at order 0 and 1, reducing memory cost significantly.

We evaluate the memory cost ratio and execution over-

head ratio of our detector. The memory cost ratio is defined as the ratio of the memory cost of our detector to the total run-time memory usage¹ without the detector. The execution overhead ratio is defined as the ratio of the increased time cost by our detector to the execution time without the detector. As shown in Table 3, the memory footprint is increased by less than 15% under our detector (except for Vortex and Eddy) and only 1-5% for majority of applications. Such a small memory cost is due to the fact that the memory footprints occupied by the applications themselves are significantly larger than the memory sizes cost by the state variables. Similarly, the huge memory cost of our detector for the Vortex and Eddy is due to the relatively large memory size occupied by the state variables. The execution overhead (including computation cost and communication cost with MPI_Allreduce) of the detector, as presented in the table, is reduced to $\leq 6.3\%$ when running all applications in a relatively large scale environment (such as 1024 cores). The diversity of the execution overhead ratio is due to the different workloads on computation and communication with various applications. It is worth noting that the execution overhead often decreases with scales (except for a few cases such as Sodshock, in which we observe a tiny increase of execution overhead with scales), indicating a relatively high scalability of our detector. The key reason is that our detector can run with the application distributively and the data prediction step in our detector is fairly lightweight on computation because of simple curve-fitting computations with little communication overhead introduced.

6 RELATED WORK

Efficient SDC detection methods have been extensively explored for years. They can be split into three categories,

¹Run-time memory usage is evaluated by Resident Set Size (RSS), which is the amount of memory occupied by the running process.

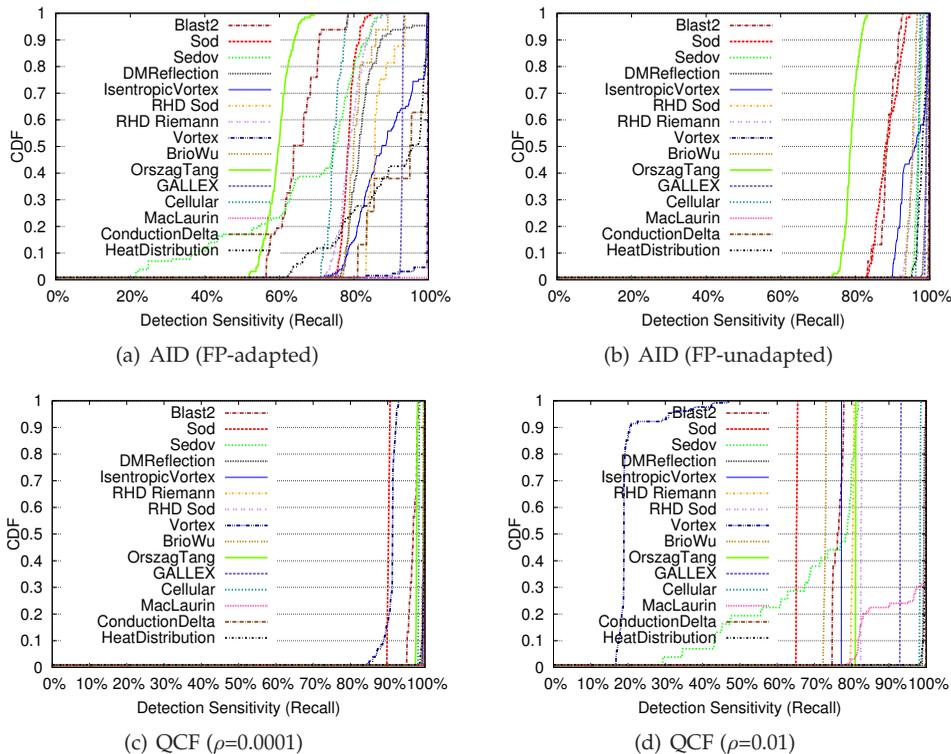


Fig. 12: CDF of Detection Sensitivity (Recall)

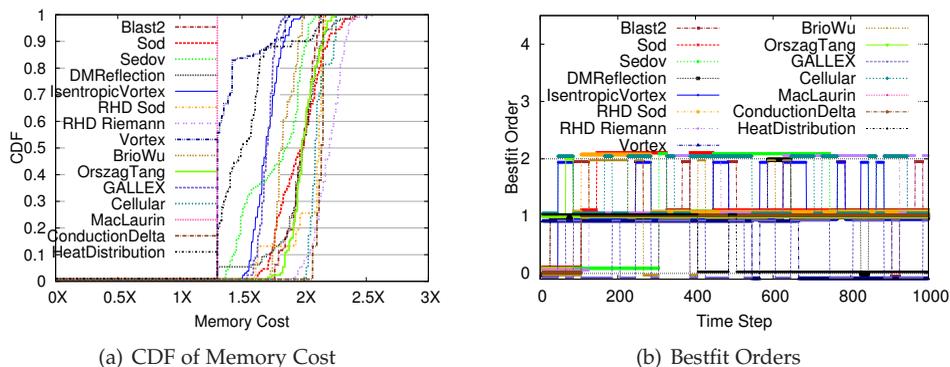


Fig. 13: Memory Cost and Best-fit Orders

TABLE 3: Analysis of Detection Overhead

Application	Memory Cost Ratio			Execution Overhead Ratio		
	256 cores	512 cores	1024 cores	256 cores	512 cores	1024 cores
Blast2	1.65%	2.1%	1.56%	3.67%	6.13%	3.93%
SodShock	0.95%	2.19%	0.44%	3.2%	3.5%	3.6%
Sedov	0.84%	0.71%	3.7%	4.6%	3.6%	3.7%
DMReflection	1.25%	1%	1.34%	3%	3.64%	3.46%
Isent.Vortex	4.7%	1%	1.5%	2.65%	1.89%	1.18%
RHD_Sod	1.6%	3.9%	4.1%	3.11%	3.4%	3.4%
RHD_Riemann	1.29%	1.27%	3.9%	4.35%	3.6%	3.7%
Eddy	25.3%	10.6%	23.9%	12%	8.6%	5%
Vortex	44%	52%	48%	1.2%	0.5%	0.4%
Briowu	9.4%	10.1%	1.9%	2.5%	3.74%	2.85%
OrszagTang	1.12%	1.98%	1.77%	2.46%	5.1%	3%
BlastBS	2.83%	0.87%	0.71%	3.1%	3%	2.98%
Cellular	2.16%	1.66%	3.38%	3.02%	3.44%	3.38%
DustCollapse	13.7%	14.6%	14.1%	1.4%	1.13%	1.12%
MacLaurin	12.8%	13%	9.15%	1.55%	1%	1.3%
Condu.Delta	1.17%	3.84%	1.8%	3.1%	3.6%	2.97%
HeatDist.	0.9%	0.72%	0.85%	13.5%	8.5%	6.3%

replica based detection, algorithm-based fault tolerance (ABFT) and runtime data analysis based detection.

The replica based detector creates the replica processes or messages to detect and correct possible errors. A typical example is RedMPI [46], which creates “replica” MPI tasks and performs online MPI message verification intrinsic to

existing MPI communication. The drawback is that it leads to relatively high redundancy of resources.

ABFT [41], [42], [43], [44], [14], [15] is a type of customized solution that protects against soft errors (e.g., SDC faults) based on the fundamental analysis of linear algebra/matrix operations (e.g., sparse linear algebra [45]). Such approaches could work effectively/efficiently in detecting silent corruptions with well-limited overhead. However, they are fairly specific to particular kernels and applications, thus they cannot be used by all HPC applications. Instead, we propose a generic detector that takes advantage of dynamic runtime data analysis during the execution.

The detection methods based on runtime data analysis have also been extensively studied recently. Yim [7] proposed an approach (called VHED) for protecting the HPC applications against transient faults on GPU devices. He designed a bin-structure-based algorithm for detecting the anomalies that appear noticeably different from other data values. Fiala et al. [46] proposed a tunable, software-based DRAM error detection and correction library (named

LIBSDC) for HPC. LIBSDC works by analyzing the overall memory pages at runtime, so it cannot selectively protect the sensible variables, which may definitely suffer from high detection overhead. Chalermarrewong et al. [47] proposed a time-series-based method to predict failures of data centers. Their solution is designed for data centers instead of HPC applications, so it cannot be directly used to detect SDC errors. Moreover, it adopts a supervised learning method, auto-regressive moving algorithm (ARMA), which requires a training period to construct the estimate coefficients, suffering extra detection overhead.

In our previous work [2], [3], [8], we proposed a one-step ahead prediction model for detecting the SDC errors. In those studies, we compared different prediction methods (such as LCF, QCF, AR, and ARMA) by using about five different applications. By comparison, we present three new contributions in this work. (1) We comprehensively analyze the HPC data features based on up to 18 real-world applications. (2) We carefully investigate the impact of SDC on the execution results, so as to explore the appropriate bound of the detection range for our impact-driven detector. (3) We design a novel adaptive detector that can reduce the false alarms and memory cost significantly, with user-acceptable execution results based on our analysis of SDC's impact.

7 CONCLUSION AND FUTURE WORK

We designed a novel adaptive impact-driven detector to protect HPC applications against influential SDCs. We characterized the HPC data features and the impact of SDC on the results by using 18 real-world applications across from different domains. Our detector selects best-fit prediction methods based on local runtime data and automatically tunes the detection range upon false positive events. The key findings are listed below.

- For the time series data of HPC applications, the data are smooth overall in the whole period, while sharp data changes exist at some time steps. This indicates that the one-step ahead prediction method should work effectively for detecting SDCs at runtime, while it is necessary to devise an adaptive method that can suit different data features.
- The characterization of SDC impact on 18 real-world applications indicates that: when the impact error bound ratio θ is set to 0.00078125, the impact of SDC can be limited below 2% of the data value range in the whole execution period for most of the cases (an exception is Blast2, whose bound ratio recommended is 0.0001). Thus, $\theta=0.00078125$ or 0.0001 is a recommended bound ratio.
- Through the experiments with the real-world applications running on up to 1024 cores from Argonne FUSION cluster, our detector can detect 80-99.99% of influential SDCs, with the false positive rate reduced to 0-1% in most cases compared with 10-99% under other state-of-the-art approaches.
- The memory cost can be reduced by 37.5-67.5% than other state-of-the-art solutions, with real memory footprint overhead reduced to $\leq 15\%$ and execution overhead reduced to $\leq 6.3\%$ for a large majority of cases studied in this paper.

In future work, we will investigate how to protect the applications (such as Eddy and BlastBS), in which the impacts of SDCs on the execution results cannot be characterized based on relative data change ratio. We also plan to study how to automatically optimize the values of the parameters used in the detector (such as θ and λ) during the execution for further improving the detection ability.

ACKNOWLEDGMENTS

This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research Program, under Contract DE-AC02-06CH11357; and by the ANR RESCUE, the INRIA-Illinois-ANL-BSC Joint Laboratory on Extreme Scale Computing, and Center for Exascale Simulation of Advanced Reactors (CESAR) at Argonne.

REFERENCES

- [1] M. Snir, et al, "Addressing failures in exascale computing," in *International Journal of High Performance Computing Applications*, pages 1- 45, 2014.
- [2] S. Di, E. Berrocal, L. Bautista-Gomez, K. Heisey, R. Gupta, and F. Cappello, "Toward Effective Detection of Silent Data Corruptions for HPC Applications," ser. SC'14 – poster section.
- [3] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, and F. Cappello, "Lightweight Silent Data Corruption Detection Based on Runtime Data Analysis for HPC Applications," tech. report. [online]. Available: <http://www.mcs.anl.gov/publication/lightweight-silent-data-corruption-detection-based-runtime-data-analysis-hpc>
- [4] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, and F. Cappello, "Lightweight Silent Data Corruption Detection Based on Runtime Data Analysis for HPC Applications," in *proceedings of High-Performance Parallel and Distributed Computing (HPDC'15)*, 2015.
- [5] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, and R. Brightwell, "Detection and Correction of Silent Data Corruption for Large-Scale High-Performance Computing," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*, Los Alamitos, CA, USA: IEEE Computer Society Press, pages 78:1-78:12, 2012.
- [6] A. R. Benson, S. Schmit, and R. Schreiber, "Silent Error Detection in Numerical Time-Stepping Schemes," in *International Journal of High Performance Computing Applications*, pages 1-23, 2014.
- [7] K. S. Yim, "Characterization of Impact of Transient Faults and Detection of Data Corruption Errors in Large-Scale N-Body Programs Using Graphics Processing Units," in *Proceedings of 28th International Parallel and Distributed Processing Symposium (IPDPS'14)*, pages 458-467, 2014.
- [8] S. Di, E. Berrocal, F. Cappello, "An Efficient Silent Data Corruption Detection Method with Error-Feedback Control and Even Sampling for HPC Applications," in *Proceedings of 15th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CC-Grid'15)*, 2015.
- [9] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, and N. Maruyama, S. Matsuoaka, "FTI: High Performance Fault Tolerance Interface for Hybrid Systems," in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*, pages 32:1-32:32, 2011.
- [10] MPICH library: [online] <http://www.mpich.org/>
- [11] S. Di, M.-Slim Bouguerra, L.A. Bautista-Gomez, and F. Cappello, "Optimization of Multi-level Checkpoint Model for Large Scale HPC Applications," in *Proceedings of 28th International Parallel and Distributed Processing Symposium (IPDPS'14)*, pages 1181-1190, 2014.
- [12] S. Di, L.A. Bautista-Gomez, and F. Cappello, "Optimization of a Multilevel Checkpoint Model with Uncertain Execution Scales," in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14)*, pages 907-918, 2014.
- [13] G. Bosilca, A. Bouteiller, T. Héroult, Y. Robert and J. Dongarra, "Composing resilience techniques: ABFT, periodic and incremental checkpointing," in *International Journal of Networking and Computing (IJNC)*, 5(1): 2-25, 2015.

- [14] D. Li, Z. Chen, P. Wu, and J. S. Vetter, "Rethinking Algorithm-based Fault Tolerance with A Cooperative Software-Hardware Approach," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13)*. ACM, New York, NY, USA, 44:1 - 44:12, 2013.
- [15] Z. Chen, "Online-ABFT: An Online Algorithm Based Fault Tolerance Scheme for Sot Error Detection in Iterative Methods," in *18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'13)*, pages 167-176, 2013.
- [16] ASCF Center. FLASH User's Guide (Version 4.2). [online] Available at http://flash.uchicago.edu/site/flashcode/user_support/flash2_users_guide/docs/FLASH2.5/flash2_ug.pdf
- [17] V. Eswaran and S. B. Pope, "An examination of Forcing in Direct Numerical Simulations of Turbulence," in *Computers and Fluids*, 16:257-278, 1988.
- [18] P. Colella and P. R. Woodward, "The Piecewise Parabolic Method (PPM) for Gas-Dynamical Simulations," in *Journal of Computational Physics (JCP)*, 54:174-201, 1984.
- [19] G. A. Sod, "A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws," in *Journal of Computational Physics (JCP)*, 27:1-31, 1978.
- [20] L. I. Sedov, "Similarity and Dimensional Methods in Mechanics (10th Edition)," New York: Academic, 1959.
- [21] H. C. Yee, M. Vinokur, M. J. Djomehri, "Entropy splitting and numerical dissipation," in *Journal of Computational Physics (JCP)*, 162:33-81, 2000.
- [22] J. M. Marti and E. Muller, "Numerical hydrodynamics in special relativity," in *Living Rev. Relativ.*, 6:7, 2003.
- [23] C. W. Schulz-rinne, J. P. Collins, and H. M. Glaz, "Numerical Solution of the Riemann Problem for Two-dimensional Gas Dynamics," in *SIAM J. Sci. Comput.*, 14:1394-1414, 1993.
- [24] O. Walsh, "Eddy Solutions of the Navier-Stokes Equations," in *Proceedings of The Navier-Stokes Equations II - Theory and Numerical Methods*, 306-309, Oberwolfach 1991.
- [25] P. Fisher, "Nek5000 User Guide," [online] Available at <http://www.mcs.anl.gov/fischer/nek5000/examples.pdf>.
- [26] P. J. O'Rourke and M. S. Sahota, "A Variable Explicit/Implicit Numerical Method for Calculating Advection on Unstructured Meshes," in *Journal of Computational Physics (JCP)*, 143:312-345, 1998.
- [27] M. Brio and C. C. Wu, "An Upwind Differencing Scheme for the Equations of Ideal Magnetohydrodynamics," in *Journal of Computational Physics*, 75:400-422, 1988.
- [28] S. A. Orszag and C. M. Tang, "Small-Scale Structure of Two-Dimensional Magnetohydrodynamic Turbulence," in *Journal of Fluid Mechanics (JFM)*, 90:129-143, 1979.
- [29] A. L. Zachary, A. Malagoli, and P. Colella, "A Higher-Order Godunov Method for Multidimensional Ideal Magnetohydrodynamics," in *SIAM Journal of Scientific Computing*, 15(2):263-284, 1994.
- [30] A. Obabko. Simulation of Gallium Experiment. [online] Available at: <http://www.cmso.info/cmsopdf/princeton5oct05/talks/Obabko-05.ppt>
- [31] F. X. Timmes, M. Zingale, K. Olson, B. Fryxell, P. Ricker, A. C. Calder, L. J. Dursi, H. Tufo, P. MacNeice, J. W. Truran, and R. Rosner, "On the Cellular Structure of Carbon Detonations," in *Astrophysical Journal*, 543:938-954, 2000.
- [32] S. A. Colgate and R. H. White, "The Hydrodynamic Behavior of Supernovae Explosions," in *The Astrophysical Journal*. 143:626-681, 1966.
- [33] N. Ozisik, "Finite Difference Methods in Heat Transfer," *CRC Process*, 1994.
- [34] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, p. MacNeice, R. Rosner, J. W. Truran, and H. Tufo, "Flash: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes," in *The Astrophysical Journal Supplement Series (ApJS)*, 131:273-334, 2000.
- [35] Nek5000 project. [online]. Available : <https://nek5000.mcs.anl.gov>
- [36] Fusion Cluster. [online]. Available at : <http://www.lcr.anl.gov/>
- [37] R. H. Shumway, "Applied Statistical Time Series Analysis," *Prentice Hall*, Englewood Cliffs, NJ, 1988.
- [38] S. Saini, P. Mehrotra, K. Taylor, S. Shende, R. Biswas, "Performance Analysis of Scientific and Engineering Applications Using MPIinside and TAU," in *Proceedings of 12th IEEE International Conference on High Performance Computing and Communications (HPCC'10)*, pages 265-272, 2010.
- [39] S. Kumar and D. Faraj, "Optimization of MPI Allreduce on the Blue Gene/Q supercomputer," in *Proceedings of the 20th European MPI Users' Group Meeting (EuroMPI'13)*, pages 97-103, 2013.
- [40] AID SDC detection library. [online]. Available at: <https://collab.mcs.anl.gov/display/ESR/AID>.
- [41] M. Turmon, R. Granat, D. Katz, J. Lou, "Tests and Tolerances for High-Performance Software-Implemented Fault Detection," in *Transactions on Computers*, 52(5):579-591, 2003.
- [42] J. A. Gunnels, D. S. Katz, E. S. Quintana-Orti, and R. A. van de Geijn, "Fault-Tolerant High-Performance Matrix Multiplication: Theory and Practice," in *Proceedings of International Conference on Dependable Systems and Networks (DSN'00)*, pages 47-56, 2001.
- [43] M. Turmon, R. Granat, and D. S. Katz, "Software-Implemented Fault Detection for High-Performance Space Applications," in *Proceedings of International Conference on Dependable Systems and Networks (DSN'00)*, pages 107-116. 2000.
- [44] E. Ciocca, I. Koren, Z. Koren, C. M. Krishna, and D. S. Katz, "Application-Level Fault Tolerance and Detection in the Orbital Thermal Imaging Spectrometer," in *Proceedings of Pacific Rim International Symposium on Dependable Computing*, pages 43-48, 2004.
- [45] J. Sloan, R. Kumar, and G. Bronevetsky, "Algorithmic Approaches to Low Overhead Fault Detection for Sparse Linear Algebra," in *Proceedings of International Conference on Dependable Systems and Networks (DSN'12)*, pages 1-12, 2012.
- [46] D. Fiala, K. B. Ferreira, F. Mueller, and C. Engelmann, "A Tunable, Software-based DRAM Error Detection and Correction Library for HPC," in *Proceedings of International Conference on Parallel Processing (Euro-Par'11)*, pages 251-261, 2011.
- [47] T. Chalermarwong, T. Achalakul, and S. C. W. See, "Failure Prediction of Data Centers Using Time Series and Fault Tree Analysis," in *18th Proceedings of International Conference on Parallel and Distributed Systems (ICPADS'12)*, pages 794-799. 2012.



Sheng Di Sheng Di received his master degree from Huazhong University of Science and Technology in 2007 and Ph.D degree from The University of Hong Kong in 2011. He is currently a postdoc researcher at Argonne National Laboratory (ANL). Dr. Di's research interest involves resilience, high performance computing (HPC) and cloud computing. He is working on multiple HPC projects, such as detection of silent data corruption and characterization of failures and faults for HPC systems. Contact him at sdi1@anl.gov.



Franck Cappello Franck Cappello is Program Manager and Senior Computer Scientist at ANL. Before moving to ANL, he held a joint position at Inria and University of Illinois at Urbana Champaign where he initiated and co-directed from 2009 the INRIA-Illinois-ANL Joint Laboratory on Petascale Computing. Until 2008, he led a team at INRIA where he initiated the XtremWeb (Desktop Grid) and MPICH-V (fault-tolerant MPI) projects. From 2003 to 2008, he initiated and directed the Grid5000 project, a nationwide computer science platform for research in large-scale distributed systems. He has authored papers in the domains of fault tolerance, high-performance computing, Grids and contributed to more than 70 program committees. He is editorial board member of the international Journal on Grid Computing, Journal of Grid and Utility Computing and Journal of Cluster Computing. He was the Program program co-chair for ACM HPDC2014, Test of time award chair for IEEE/ACM SC13, Tutorial co-Chair of IEEE/ACM SC12, Technical papers co-chair at IEEE/ACM SC11, Program chair of HiPC2011, program cochair of IEEE CCGRID 2009, Program Area co-chair IEEE/ACM SC'09, General Chair of IEEE HPDC 2006. Contact him at cappello@mcs.anl.gov.

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.