MOX-Report No. 21/2020

Resilience and fault-tolerance in high-performance computing for numerical weather and climate prediction

Benacchio, T.; Bonaventura, L.; Altenbernd, M.; Cantwell, C.D.; Düben, P.D.; Gillard, M.; Giraud, L.; Göddeke, D.; Raffin, E.; Teranishi, K.; Wedi, N.

MOX, Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

mox-dmat@polimi.it http://mox.polimi.it

Resilience and fault-tolerance in high-performance computing for numerical weather and climate prediction

Tommaso Benacchio[(1)], Luca Bonaventura[(1)], Mirco Altenbernd[(2)], Chris D. Cantwell[(3)], Peter D. Düben[(4,9)], Mike Gillard[(5)], Luc Giraud[(6)], Dominik Göddeke[(2)], Erwan Raffin[(7)], Keita Teranishi[(8)], Nils Wedi[(4)]

[(1)] MOX – Modelling and Scientific Computing,
Dipartimento di Matematica, Politecnico di Milano, Milan, Italy
{tommaso.benacchio, luca.bonaventura}@polimi.it

[(2)] Institute for Applied Analysis and Numerical Simulation and
Cluster of Excellence 'Data-driven Simulation Science'
University of Stuttgart, Stuttgart, Germany
{mirco.altenbernd, dominik.goeddeke}@ians.uni-stuttgart.de

[(3)] Imperial College London, UK
c.cantwell@imperial.ac.uk

[(4)] European Centre for Medium Range Weather Forecasts, Reading, UK
{peter.dueben, nils.wedi}@ecmwf.int

[(5)] Loughborough University, UK
m.gillard@lboro.ac.uk

[(6)] HiePACS, Inria Bordeaux, Sud-Ouest, Talence, France
luc.giraud@inria.fr

[(7)] CEPP - Center for Excellence in Performance Programming, Atos Bull,
Rennes, France
erwan.raffin@atos.net

[(8)] Sandia National Laboratories, Livermore, CA, USA
knteran@sandia.gov

[(9)] AOPP, Department of Physics, University of Oxford, Oxford, UK

Keywords: Fault-tolerant computing; high-performance computing; numerical weather prediction; iterative solvers

AMS Subject Classification: 65-04, 65F10, 65M55, 65N22, 65Y05, 86A10, 94B99

Abstract

Numerical weather and climate prediction rates as one of the scientific applications whose accuracy improvements greatly depend on the growth of the available computing power. As the number of cores in top computing facilities pushes into the millions, increasing average frequency of hardware and software failures forces users to review their algorithms and systems in order to protect simulations from breakdown. This report surveys approaches for fault-tolerance in numerical algorithms and system resilience in parallel simulations from the perspective of numerical weather and climate prediction systems. A selection of existing strategies is analyzed, featuring interpolation-restart and compressed checkpointing for the numerics, in-memory checkpointing, ULFM- and backup-based methods for the systems. Numerical examples showcase the performance of the techniques in addressing faults, with particular emphasis on iterative solvers for linear systems, a staple of atmospheric fluid flow solvers. The potential impact of these strategies is discussed in relation to current development of numerical weather prediction algorithms and systems towards the exascale. Trade-offs between performance, efficiency and effectiveness of resiliency strategies are analyzed and some recommendations outlined for future developments.

1 Introduction

Since the dawn of computing, numerical weather prediction (NWP) and climate studies have served as a key test bed for performance of cutting-edge hardware architectures. In operational weather services, supercomputers provide the infrastructure for a round-the-clock enterprise relying on the timely execution of the forecast suite, ranging from assimilation and handling of observations that generate initial conditions to extended-range predictions.

Up to a decade ago, improvement in atmospheric fluid flow simulations followed steady increases in clock speed of supercomputers' processors. This growth, largely due to Moore's law, has flattened out in the last decade, so that, besides systematic exploitation of alternative architectures such as GPUs, increase in sheer core count represents the main factor in the current advances in computational performance. Extrapolation of the core count and performance of the machines included in the TOP500 list shows that the one-million core and exaflop (10^{18} double precision floating-point operations per second) limits are bound to be crossed in the next few years [98].

As of November 2019, the TOP500 list features at least 20 high-performance computing (HPC) systems exclusively devoted to numerical weather and climate prediction, with their combined figures exceeding two million cores, 68 Petaflops maximum LINPACK performance and memory in the region of petabytes (Table 1). This excludes general-purpose HPC systems - some very high-ranking - on which atmospheric simulations also run routinely. The growing core count and computational performance have enabled a steady increase in model resolution and, ultimately, forecast accuracy [14, 129], but have also posed serious challenges to existing modelling systems, numerical algorithms, interplay between different modelling components, computational infrastructure, memory management, efficiency, scalability, and energy consumption.

As pointed out in [109], the challenges of exascale scientific computing will revolve around *power* (current flops/watt rates project to unsustainable consumption in the range of hundreds of megawatts), *concurrency* (reduction of communications in parallel programs), *data locality* (redefining performance from flops to bandwidth/latency), and *memory* (as available memory per core is decreasing). One relatively less considered issue on the path towards exascale simulations in numerical weather prediction concerns the reliability of computing systems. While reliability was a major concern in the early days of computing (see, e.g., the discussion in [96]), the practical irrelevance of computing hardware faults has now been taken for granted

Table 1: Subset of the Top500 list (November 2019 standings) with ranking, owning institution, core count (in thousands), and maximum LINPACK performance Rmax for systems dedicated to numerical weather prediction and climate studies, operational or for research (for more details see [98]).

Ranking [/500]	Institution	KCores	Rmax [PFLOPs]
27,87,88	United Kingdom Meteorological Office	241.9 + 89.86(2)	7.039+2.802(2)
33,34	Japan Meteorological Agency	135.79(2)	5.731(2)
44,339	NCAR (US)	144.9+72.29	4.788+1.258
54,55	ECMWF	126.47(2)	3.945(2)
57	Indian Institute of Tropical Meteorology	119.32	3.764
80	Deutsches Klimarechenzentrum	99.072	3.011
102,110	China Meteorological Administration	50.82+48.13	2.547+2.435
100	NCMRWF (India)	83.59	2.570
113,114	Korea Meteorological Administration	69.6(2)	2.396(2)
127,132	Météo-France	72+73.44	2.168+2.157
341,342	NOAA (US)	48.96(2)	1.635(2)
345	Beijing Meteorological Association	28.8	1.624
460	Deutscher Wetterdienst	41.47	1.214
465	Roshydromet (Russia)	35.14	1.200

for many years. However, infallibility assumptions for HPC hardware, as well as absence of faults at the bit level, inevitably cease to hold as the number of processors exponentially expands and their size shrinks. Inadvertently, applications spread across a larger number of computational nodes with single point of failures. Bit-level faults may also be exaggerated in the future due to a trend to exploit reduced precision in order to accelerate the time-to-solution [54].

1.1 Numerical weather and climate prediction models

An atmospheric prediction system takes input data obtained through complex *data assimilation* procedures - variational optimization problems involving observations and previous model output - and uses them as initial conditions for a numerical model approximating the physical laws governing the evolution of atmospheric flow.

Within the numerical model, the *dynamical core* is responsible for the simulation of the atmospheric flows resolved by the computational grid, usually discretizing the inviscid, rotating, compressible Navier-Stokes equations under the action of gravity. The typical prognostic variables include wind, air density or pressure, as well as thermodynamic and water substance variables – different options for variables and equation sets are discussed, e.g., in [142]. The numerical solution involves handling the transport by the wind on the one hand and the faster dynamics of sound waves and internal gravity waves, induced by the compressibility and vertical stratification of the atmosphere, on the other [18, 19, 134, 29]. Semi-implicit time discretization strategies are widely employed in atmospheric models in order to achieve efficiency by employing large time steps unconstrained by the speed of fast waves. Semi-implicit approaches, however, imply the formulation and solution of global linear systems that make model discretizations more involved and parallelization harder than with explicit methods. Experience with operational semi-implicit discretizations suggests that linear solvers routinely take up sizeable portions of wall-clock time in model runs (see, e.g., Figure 2 in [104]). In this context, particular attention has been directed to the choice and construction of suitable preconditioners for these systems. Conceptually similar linear solvers also feature in the data assimilation process, to which analogous considerations apply. Indeed, data assimilation routinely takes up the largest share of the total computational time required to produce a weather forecast, and its resilience is therefore of paramount importance for the timely dissemination of forecast products.

In addition to the dry inviscid dynamics solved by the dynamical core, many of the processes associated with meteorologically relevant phenomena - convection, radiation, cloud microphysics, boundary layer turbulence, gravity wave drag, and others, customarily referred to in meteorology as the *physics* - occur at scales that cannot be resolved by the computational grid, so their effects are parametrized and fed into the dynamical core as source terms. Model complexity is further enhanced by other components, such as the ocean and wave model, the land surface model, the atmospheric chemistry model, and their mutual coupling.

Today's state-of-the-art global operational NWP systems provide forecasts simulated with 25-9 km average grid-spacing in the horizontal direction. Limited area models use grid-spacings down to 1 km. Ensembles with perturbed initial conditions and additional model perturbations with up to 50 members are used to sample the initial condition and model uncertainty. The vertical discretization uses approximately 100 unevenly spaced vertical levels out to model tops of around 80 km. This translates into around 500 million spatial degrees of freedom per variable, requiring a couple of thousand forward time steps for a two-week forecast.

Especially in NWP, due to the relatively short window between the arrival of observations, the quality control, assimilation and subsequent dissemination of the forecast, typically only one hour is available for the two-week forecast including uncertainty estimation. Efficiency and in particular scalability, with a mixture of both strong scaling and weak scaling requirements, become paramount in a context of expanding massively parallel resources. Recent studies [56, 105, 129] warned that current model efficiency needed to improve by at least two orders of magnitude in order to be able to run timely 1 km global NWP simulations on exascale systems. As a result, failed tasks or other hardware issues can very quickly lead to delayed forecast dissemination. Thus, strategies for resilience such as hosting identical twin computing clusters for instant switching of workloads, separating research and operational (peak) resources, and checkpointing intermediate forecast results to disk are already common practice.

1.2 Computational challenges and aims of the study

The present paper aims to explore the reliability issue in numerical weather and climate prediction applications by:

- establishing a reference taxonomy of faults, errors, and failures;

- surveying the traditional approaches to reliability in computing, and considering their shortcomings;
- exploring data recovery strategies used by current research on fault-tolerance for linear solvers and on systems' hardware resilience;
- presenting example applications of these approaches to NWP demonstrators;
- discussing the resilience/efficiency trade-off and admissible overheads with a view to improving existing NWP systems' reliability.

In the following, resilience will be used as an umbrella term for techniques that keep applications running despite faults. Most likely this will be accomplished by combining hardware, programming environment and application resilience measures. Because of the nature of solvers used in NWP, data recovery approaches that are *local* in nature will be of particular interest given the increasing communication/computation time ratio in discrete models.

1.3 Taxonomy and traditional approaches

In the reliability literature, *faults* are defined as causes of *errors*, which are parts of the state causing a failure. The latter represents the transition to incorrect service (for taxonomy and wider related literature we refer to the reviews in [11, 34, 35, 49, 102, 109, 133]). Parallel computing systems may undergo hardware breakdowns in several of their components - processes, nodes, blades, cabinets. These breakdowns are usually defined as *hard faults*, they are in general reproducible and, if unaddressed, bring programmes to halt. *Soft faults* are instead usually caused by fluctuations in radiation that introduce spurious modifications in the programme data in the form of *bit flips* and are usually non-reproducible [38]. Faults are further distinguished into detected and corrected, detected and uncorrectable, and undetected ones. The latter category can take the form of *silent data corruption* and lead the programme to compute the wrong solution unbeknown to the user [32, 44, 57, 58, 59, 64, 65, 76, 92, 99]. Next to other performance figures in HPC, mean time between failure (MTBF), made of the sum of mean time to interrupt (MTTI) and mean time to repair (MTTR), has arisen as a measure of reliability of a computing system.

The aforementioned exponential growth in the core counts of high-performance computing systems carries the potential to shrink fault-free computing cycles, dramatically so in some cases. The

authors of [69] report a MTBF of 9-12 hours. Lower figures were identified for Blue Waters [48] and the Tianhe-2 machines [37]. The authors of [77] present the normalized MTBF of 5 systems over 8 years (their Figure 1). As reported in [41], the MTBF of a system with one million nodes, each of which with a MTBF of 3 years, drops to about 94 seconds.

Published figures in machines used in weather centres are generally hard to come by. The two supercomputers at the European Centre for Medium-Range Weather Forecasts (ECMWF), with a total 7220 Cray XC-40 nodes and a joint peak performance of more than 8 petaflops, show about 15 node failures per months, including memory failures, CPU failures and software crashes but excluding preventive maintenance. (Christian Weihrauch, personal communication). See also [12, 17, 77, 128] for more experimental resilience assessments.

Efforts to limit the impact of computing failures have usually revolved around *rollback* strategies, mostly in the form of *checkpoint-restart* (CPR). In this approach, state data are written out to a parallel filesystem, enabling the application to continue from this consistent state after being restarted following a failure. Checkpointing is intuitive in principle and is commonly agreed to be good programming practice for code bases of a certain size. However, it is computationally expensive as it involves moving data to and from disk, a potential requeuing of the job when run under batch submission systems, and an increase in total system resource and energy usage. Looking ahead, the delays in time-to-solution and overhead caused by CPR may become less and less sustainable for time-bound applications, given the inherently increasing failure frequency in future operating schedules. While studies have identified optimal checkpointing intervals [41], CPR clearly stops being worthwhile at the point when the checkpointing procedure takes longer than the average mean time between failure that it is meant to protect from. As an example, a study [133] reports CPR times of 2000 s for a 64 TB dump on 1000 nodes.

Other traditional resilience strategies have used other forms of redundancy, typically replication [20, 49]. While these approaches may present some advantages when MTBF is low and CPR is heavy, they also carry sizeable overheads and are generally power-hungry.

To protect the system from bitflips, error-correcting code memory (ECC) is almost exclusively used in computing centres. The associated overhead in terms of performance, storage, and power consumption is not negligible. Depending on the configuration, capacity overhead varies between 10 % and 40 % [91] and performance-

power overhead around 10–20 % [86]. In the hardware community, a 12.5 % overhead appears to be a commonly accepted value [135]. In this context, figures released by hardware vendors tend to include performance penalty only, overlooking the power requirements for ECC/Chipkill[45]. Memory(DRAM) protection - increasing refresh rate and adding more parity modules/bits to ECC/Chipkill - also needs energy. The hardware community agreed on the power or performance penalty associated with reliability enhancement. Despite new techniques enabling more efficient reliability enhancement, it is inevitable for computer system architects to allocate some power and performance budget for reliability. For example, in [85] a highly reliable Chipkill was implemented with two extra chips for a 16-chip memory module. Other authors suggested hybridizing dynamic refresh rates and Chipkill together to tune reliability, power and performance tradeoffs.

At any rate, high-performance computing hardware could work much more efficiently if the constraint to always calculate the exact answer could be weakened. A number of studies recently investigated the possibility to reduce numerical precision in weather simulations to allow for a reduction in computing cost. Savings can be reinvested to increase model resolution or the number of ensemble members for ensemble forecasts to improve predictions. Studies have covered atmosphere models [55] but also data assimilation [79] and other model components such as land surface [43]. It has also been argued that reduced numerical precision may not necessarily degrade results of simulations for weather predictions. In contrast, in some cases it may even be possible to use variability from rounding errors to improve ensemble simulations [52].

A possible reduction in numerical precision has been investigated using *stochastic processors* [54] and exact arithmetic, for example employing programmable hardware such as Field Programmable Gate Arrays (FPGAs) [122]. Stochastic processors reduce power consumption through voltage over scaling. Here, the voltage applied to the processor is reduced beyond the level at which all operations are calculated correctly. Results suggest that power consumption could indeed be reduced significantly if a low number of faults is acceptable for the applications [53, 94, 127].

For all these reasons, even though HPC hardware reliability has been relatively stable in recent years, resilience to hard and soft faults for NWP models should now be investigated more thoroughly, given increasingly tighter production schedules and expanding core counts. Scalability tests with weather models using a significant fraction of some of the world's fastest supercomputers are still feasible

without serious problems due to faults [56, 66], but checkpointing to disk is already required. As an example, the operational deterministic global forecast at ECMWF at 9 km resolution runs with 324 compute nodes and 28 I/O server nodes with 4 MPI tasks per node. This translates into 202MB of data to be written for a restart file per MPI task – including the models for atmosphere, ocean, waves and ice. Writing a restart dump for this system takes one to three seconds. During a 10-day forecast, five checkpoints are written, requiring less than 1% of the total forecast runtime. Although these figures suggest that there is still some leeway before CPR becomes unaffordable in operational NWP, checkpointing already absorbs almost the entire bandwidth of the Lustre file-system. At 1km resolution for the atmospheric model component, the total size of restart files reaches 4 Terabytes, which may be simultaneously written from across thousands of MPI tasks to a single filesystem. This raises questions regarding the interference of different users jobs on the entire HPC performance.

1.4 Modern approaches and paper outline

Given the relatively large overhead of traditional resilience techniques, in recent years computing experts have started exploring alternative, more economical approaches, exploiting also emerging non-volatile memory architectures becoming available at the compute node level. For convenience, here we divide these approaches in two groups, one related to algorithmic fault-tolerance models, and one related to systems' resilience.

Methods in the first group modify existing algorithms, typically iterative methods for the solution of linear systems, in order to ensure continuing execution and solution quality upon hard or soft faults. To this category belong the *interpolation-restart* (IR) algorithms for iterative methods solving sparse linear systems [2, 3, 4, 5, 6, 7] and schemes guided by the *algorithm-based fault-tolerance* approach (ABFT, see [50, 81] and [9, 74] for multigrid applications). Section 2.1 contains more details on two approaches of this kind.

Methods in the second group are based on techniques that alter parallelization schedules (in user level failure mitigation - ULFM, see [8, 10, 26, 33, 63, 70, 71, 138]) or create convenient backup systems [51] without modifying the lower-level numerical implementation. ULFM is usually associated with the actual set of MPI extensions proposed to support the identification and mitigation of failures within MPI communicators and restoring those communicators back to working condition so the application may continue, see

Section 2.3, where two flavours of ULFM are explored more in depth. An approach against hard and soft faults based on a backup grid is presented in Section 2.4. Next, Section 2.5 contains some considerations on HPC hardware resilience, based on the example of the BullSequana system. In Section 3 we present preliminary numerical results obtained applying some of the described algorithms to implementations derived from numerical weather prediction codes. Wider applicability in the NWP context, included interplay with emerging domain-specific language-based software frameworks, is discussed in Section 4 and conclusions are drawn in Section 5.

Other approaches not explored in this study involve reducing the precision of calculations or exactness of operations in order to reduce the computational – and hence checkpoint-restart – load on simulations [30], employing advanced focused checkpointing strategies [40, 83, 87, 95, 113, 119, 124, 137], selectively identifying the parts of models in need of reliability [31, 80], exploiting self-stabilizing iterative solvers [125], using data compression [47], and global view resilience [39]. Resilience in the framework of domain decomposition preconditioners is explored in [117, 118, 126]. We refer to the cited studies for further discussion on these techniques. We also remark that we only deal with fault recovery techniques, while discussion of fault detection is limited, for more on that see, e.g., [15, 22, 23], and [141] for failure prediction.

2 Software resilience

Algorithmic fault-tolerance approaches aim at supplying numerical methods with techniques to deal with hard or soft faults. In general, approaches of this kind modify the theoretical formulation and the implementation of the standard non fault-tolerant version of the methods. Some methods, for example multigrid, structurally lend themselves well to the task.

The effectiveness of these approaches can be evaluated based on the trade-off between the scope of changes needed for fault-proofing and related computational overhead on the one hand and fault-tolerance performance on the other. Of particular interest in the NWP context are fault-tolerant versions of iterative algorithms for linear systems solution. Two examples are given below.

We consider the formalism proposed in [90] where data losses are classified into three categories: *computational environment*, *static* data and *dynamic* data. For Section 2.1 and 2.2 we are only dealing with dynamic data recovery.

2.1 Interpolation-restart

Interpolation-Restart techniques are designed to cope with node crashes (hard faults) in a parallel distributed environment. The methods can be designed at the algebraic level for the solution both of linear systems and of eigenvalue problems.

In the framework of iterative methods for the solution of linear algebra problems, we consider numerical algorithms that are able to extract relevant information from available data after a fault, assuming a separate mechanism for fault detection. After data extraction, a well-chosen part of the missing data is regenerated through interpolation strategies, and is then used as meaningful input to restart the iterative scheme.

A well-suited initial guess is computed by using the entries of the faulty iterate available on surviving nodes. Two fully algebraic interpolation policies can be considered, that require the solution of either a small linear system or a small least-squares problem. The schemes are designed at application level and require no extra computational units or computing time when no fault occurs.

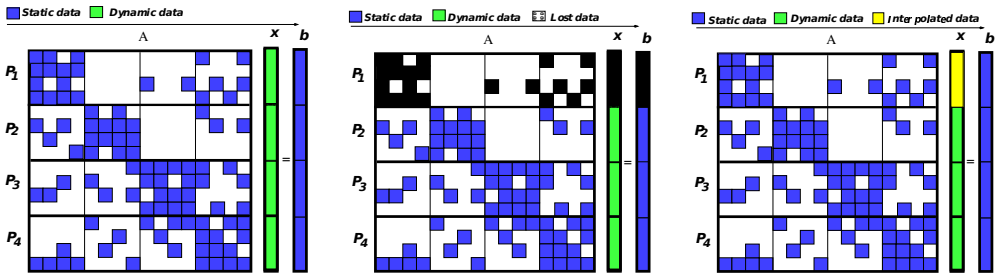
Let us briefly describe the basic idea in the context of a single fixed-point Richardson iteration scheme (see e.g. [115]) for the solution of a linear system

$$Ax = b \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$ denotes a nonsingular matrix, $b \in \mathbb{R}^n$ the right hand side and $x \in \mathbb{R}^n$ the solution. Without loss of generality, we assume that all the vectors or matrices of dimension n are distributed by blocks of rows in the memory of the different computing nodes. Consistently, the iterate $x^{(k)}$, the right-hand side b and the coefficient matrix A are distributed according to a block-row partition.

Let N be the number of partitions, such that each block-row is mapped to a computing node. For all p, $p \in [1, N]$, I_p denotes the set of row indices mapped to node p. With respect to this notation, node p stores the block-row $A_{I_p,:}$ and x_{I_p} as well as the entries of all the vectors involved in the solver associated with the corresponding row indices of this block-row. If the block A_{I_p,I_q} contains at least one nonzero entry, node p is referred to as neighbour of node q, as communication will occur between those two nodes to perform a parallel matrix-vector product.
Using an initial guess $x^{(0)}$, at step k the Richardson iteration is

$$x^{(k+1)} = x^{(k)} + \left(b - Ax^{(k)}\right). \tag{2}$$



(a) Before Fault (b) Faulty iteration (c) Interpolation

Figure 1: General interpolation scheme. The matrix is initially distributed with a block-row partition, here on four nodes (a). When a fault occurs on the node P_1, the corresponding data are lost (b). Whereas static data can be immediately restored, dynamic data that have been lost cannot. We investigate numerical strategies for regenerating them (c).

When a node crashes, all available data in its memory are lost. The computational environment consists of all the data needed to perform the computation (programme code, environment variables, etc.). Static data are set up during the initialization phase and remain unchanged during the computation. In the case of the solution of a linear system, they correspond to the input data to the problem and include in particular the coefficient matrix A and the right-hand side vector b. Dynamic data are those whose value may change during the computation, that reduces to the iterate $x^{(k)}$ for a single fixed-point iteration scheme.

Figure 1a shows a block row distribution on four nodes, with static data (matrix and right-hand side, blue) and dynamic data (the iterate, green) associated with the linear system. If node P_1 fails, the first block-row of A as well as the first entries of x and b are lost (in black in Figure 1b).

Because the primary interest here lies in the numerical behaviour of the schemes, we make strong assumptions on the parallel environment. In particular, when a fault occurs we assume that the crashed node is replaced and the associated computational environment and static data are restored [90]. This is shown in Figure 1c, where the first matrix block-row as well as the corresponding portion of the right-hand side are restored. By contrast, the dynamic iterate is permanently lost and must be recovered.

The idea consists in interpolating the lost entries of the iterate using interpolation strategies adapted to the linear systems to be solved. The interpolated entries and the current values available on the other nodes define the recovered iterate which is then used as

an initial guess to restart the Richardson iteration.

A first interpolation strategy, introduced in [90], consists in interpolating lost data by using data from surviving nodes. Let $x^{(k)}$ be the approximate solution when a fault occurs. After the fault, the entries of $x^{(k)}$ are known on all nodes except the failed one. This Linear Interpolation (LI) strategy computes a new approximate solution by solving a local linear system associated with the failed node. If node p fails, $x^{(LI)}$ is computed via

$$\begin{cases} x_{I_p}^{(LI)} = A_{I_p,I_p}^{-1}(b_{I_p} - \sum_{q \neq p} A_{I_p,I_q} x_{I_q}^{(k)}), \\ x_{I_q}^{(LI)} = x_{I_q}^{(k)} \qquad \text{for } q \neq p. \end{cases} \tag{3}$$

This basic idea can be adapted to more sophisticated linear solvers based on Krylov subspace methods [4, 90] or to eigensolvers [6] and more robust interpolation can be designed to tackle the possible singularity of the diagonal block A_{I_p,I_p}^{-1}. We refer to Section 3.1 for a more detailed description of these ideas in the context of the GMRES method.

Finally, we note that when the linear system or eigenproblem arises from the discretization of a partial differential equation (PDE), each node of the parallel platform usually handles the unknowns associated with a subdomain. In the context of a linear system solution, the interpolation policy can be viewed as the solution of the PDE on the failed subdomain with Dirichlet boundary conditions defined by the current values on the interfaces with the neighbouring domains. In that context, more sophisticated boundary conditions can be considered as presented for example in [82].

2.2 Compressed checkpointing for iterative solvers

Compressed checkpointing is an algorithm-based fault tolerance technique. It can also be characterized as a system-resilience technique, since its efficient implementation crucially relies on ULFM, see Section 2.3, and on improvements to classical CPR. Compressed checkpointing can be used in the node loss scenario, but also in case of silent data corruption (SDC). In the following, we describe some ideas for the former, more specifically for in-memory compressed checkpoints, and refer to [9] for the SDC case. Mathematical and conceptual details can be found in [74], and more details on implementation issues in [60]. The key point is that, by combining compression techniques for backup creation with local-failure local-

recovery (LFLR, [138]) approaches for iterative solvers, e.g., multigrid preconditioners, the overhead in fault-free scenarios can be reduced significantly compared to classical CPR techniques, while still providing a speed-up for recovery in the presence of hard faults or data loss.

Compressed checkpointing has the potential to recover from hard faults on-the-fly, without any interaction from the user, as it was shown using the ULFM extension for MPI or a minimalistic handcrafted standalone feature-clone in [60]. However, focusing on the solver alone is not sufficient. A prototypical implementation in the DUNE software framework [28] demonstrates that memory footprint as well as bandwidth pressure can be significantly reduced as long as some technique like message-logging, see Section 2.3, is used to recover the problem data needed to restart an iterative solver. This implementation shows the usability of different combinations of backup and recovery strategies partly based on a local-failure local-recovery approach – on which see the next section – and lossy compression techniques.

In practice, the solver is restarted in case of data loss, with an updated initial guess based on the available data on the fault-free ranks as well as the compressed backup data on the faulty rank. The advantage in terms of minimally-invasive integration into existing codes is that it is only necessary to create backups of the iterate itself, and not of additional auxiliary vectors or iterative solver data. Recovery can be further improved by storing additional information like the search direction for the conjugate gradient method. This can be especially useful if many data losses occur. In the best case, the number of iterations until convergence of the fault-prone solver before and after the restart adds up to the same amount as in the fault-free scenario. The generation of the initial guess based on the backup data can be done in multiple ways which we describe later.

Backup It can be shown that it is feasible to avoid creating a full backup on disk, at least when the previously outlined approach is combined with in-memory checkpointing techniques, see Section 2.3, storing only a compressed version of the current iterate in the memory of a neighbouring node of the cluster. The core idea is to resort to lossy compression, initially motivated by the ABFT paradigm. Contrary to lossless compression methods, e.g., *zip*, *png*, *gif* and others, lossy compression like *mp3* or *jpeg* neglects some information to increase the compression factor. Therefore, the decompressed data differ from the input data. However, since iterative solvers and preconditioners are by definition not exact, the accuracy

loss due to compression can in fact be tolerated up to a certain degree. The loss can be qualified as benign if the compression error is smaller than the error within the solver, for example the residual error accepted by some stopping criterion, and than the model numerical truncation error. In this case, the decompressed data can actually lead to results with similar quality as data from a lossless compression method with the advantage of a smaller size.

In the following, we evaluate three techniques: zero backup, multigrid, and SZ compression [47, 93, 136]. In the zero backup technique, lost data are reinitialised with zeros. Multigrid can be interpreted as a lossy compression technique, with a number of mathematical peculiarities that need consideration [74]. Multigrid algorithms use a hierarchy of grids to solve linear systems in an asymptotically optimal way. This hierarchy can be used to restrict, i.e., lossily interpolate, the iterate from fine to coarse grids. Such lower-resolution representation of the iterate can then be stored as a compressed backup. Later, the multigrid prolongation (coarse-to-fine grid interpolation) operator is used to decompress the data. Obviously, the quality of the backup is strongly dependent on the grid where it is restricted to. This compression technique can easily be applied if a multigrid solver or preconditioner is used anyway, because the operators are readily available. Furthermore, this principle can be used in combination with other hierarchic methods.

SZ compression, on the other hand, does not use operators from a specific solver or preconditioner but can be applied to all kinds of field data, although it prefers, by construction, structured data ideally associated with a structured grid. Another important feature is that the compression accuracy can be prescribed and adapted to the situation. Unfortunately, a higher compression accuracy usually leads to a lower compression factor and higher compression time, which is crucial in terms of resilience overhead.

Recovery We suggest and evaluate three recovery techniques. The first, baseline approach mimics CPR and simply replaces the *global* iterate with its decompressed representation, independently of the compression strategy. The second approach follows the LFLR strategy and re-initializes only the *local* data lost on faulty ranks by using backup data stored on neighbouring nodes. Unlike the baseline approach, this is purely local and only needs minimal communication to receive a remotely stored backup. In particular, the recovery procedure itself does not involve the participation of other ranks.

As a worst-case fallback when the backup data are not sufficient,

we established a third *improved* recovery approach, which is still mostly local. An auxiliary problem is constructed, either by domain decomposition overlap or the operator structure, with boundary data from the neighbouring ranks, and solved iteratively with an initial guess based on the checkpoint data. This improves the quality of the recovered data similar to equation (3) in Section 2.1. This approach can nearly always restore the convergence from the fault-free scenario independently of the used backup technique, only the number of additional local iterations varies, see also [9, 82]. It can be mandatory in highly-frequent fault scenarios to maintain convergence.

2.3 System resilience

System resilience techniques take a higher-level, application-based, perspective in comparison to algorithmic resilience approaches, often with the focus on handling hard faults. The objective is to protect the entire application from permanent failures in the computing resources and enabling the application to continue, potentially on a reduced set of resources. As a consequence, system resilience often concerns more generic approaches which cater to a broader range of software.

System resilience typically focuses on overcoming faults which result in the complete loss of one or more MPI processes due to, for example, hardware failure. The most basic form of resilience is replication, whereby calculations are performed in duplicate on multiple nodes. This allows immediate continuation of the calculation in the event of a failure, but with a very inefficient use of resources during normal forward-path execution. Process replication has been explored as a simple and effective approach to resilience [75], although it is highly wasteful of resources during fault-free execution.

As noted above, CPR to stable storage is the classic system resilience methodology used to avoid complete re-execution of long-running applications in the event of faults. Application-based checkpointing is common among production codes, particularly in NWP, allowing targeted data structures to be protected without wasting resources in protecting auxiliary data which can otherwise be reconstructed. We next survey a number of recent developments that have been proposed in order to improve the efficiency of the CPR technique at the system level.

Rather than forcing a restart of the entire application, localised mitigation of the failure is preferable. One particular challenge in the context of MPI applications has been the difficulty in recover-

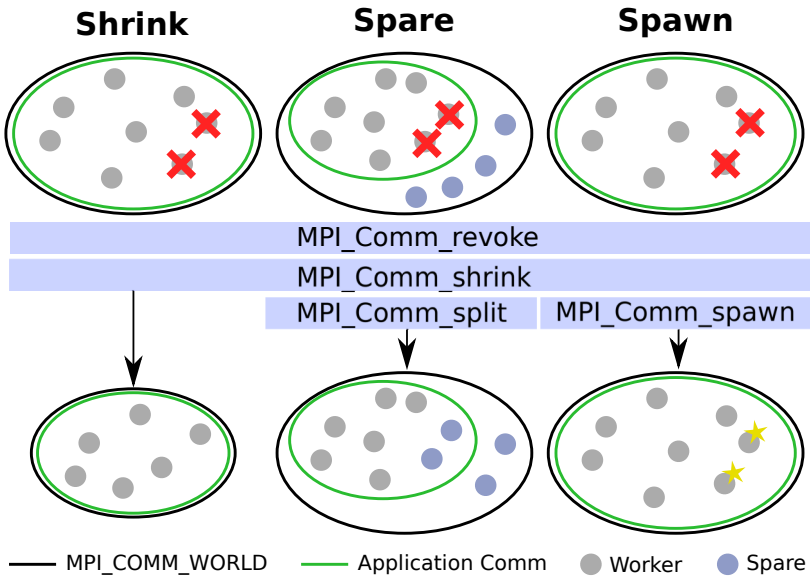


Figure 2: Approaches to communication-channel recovery using user level failure management extensions proposed for MPI 4.0.

ing MPI communication in the event of hard faults. The ULFM extensions to the MPI standard introduced in Section 1 provide an application-driven mechanism to detect and recover communication channels in the event of a hard fault leading to the loss of one or more processes. These extensions might be included in the MPI 4.0 standard [26]. In prior versions of MPI, communication routines would either trigger an immediate abortion of the program or return control to the application to support a more controlled termination. ULFM allows failed communicators to be revoked – alerting all surviving processes to the failure occurrence – and subsequently shrunk to exclude all failed processes and restore functionality of the communicator. If supported by the computational resource, additional processes may also be spawned to replace those which have failed. These strategies are illustrated in Figure 2. Although not yet part of the standard, the proposed ULFM extensions have already been applied in a number of studies [8, 10, 26, 33, 60, 63, 70, 71, 138].

Following the restoration of communicators, the application must be returned to a consistent state to continue execution, including restoration of data structures on any replacement processes. This requires mechanisms for both protection and restoration of critical data structures.

In-memory checkpointing avoids writing to the parallel filesystem. While local memory-based approaches provide excellent per-

formance for application faults, they fail to provide resilience against more serious hardware failures. Remote in-memory checkpointing places checkpoints in the memory of a remote node through a pairwise communication, retaining the performance benefits of avoiding parallel filesystem access, but at the cost of increased network traffic. Upon failure of one or more processes, remote checkpoints can either be written to disk for use with traditional CPR, or restored directly after ULFM communicator recovery for a *checkpoint-mitigate-rollback* approach.

Data transfer over the network is the main performance bottleneck of remote in-memory checkpointing. Depending on the volume of state data which must be protected per process, the volume of memory available on remote nodes for storing checkpoint data is also a concern. Increasing prevalence of fast NVRAM has addressed this issue to some extent [84, 101]. Nielsen [109] considered the application of Reed-Solomon encoding to reduce the storage requirements of checkpoints, storing only checksums in addition to the local data, at the cost of additional computation.

An alternative strategy to protecting the state of an application is to log communications between processes, thereby allowing recovery of one or more processes to be performed in isolation. Message logging avoids the need for surviving processes to rollback, but the size of message logs may grow significantly, depending on the communication pattern of the application.

Application to existing codes Mitigation of hard failures and implementation of system resilience techniques generally requires modification of existing codes to varying degrees. A number of libraries and packages already exist to facilitate this effort. Their impact ranges from almost zero intrusion through to a complete rewriting of the source code to incorporate resilience.

ACR [106] implements Automatic Checkpoint/Restart using replication of processes in order to handle both soft and hard errors. FA-MPI [78] proposes non-blocking transactional operations as an extension to the MPI standard to provide scalable failure detection, mitigation and recovery. Finally, FT-MPI [63] was a precursor to ULFM for adding fault tolerance to the MPI 1.2 standard.

These approaches, as well as Fenix detailed below, generally require significant intrusion into the application code to mark data structures to be protected and to implement the recovery process. A less intrusive approach [33], targeting time-dependent solvers, combines message-logging techniques and remote in-memory checkpointing to reduce this burden in often complex object-oriented applica-

tions. It exploits the nature of these codes by transparently logging MPI messages during the relatively short initialisation phase of the application, when the data structures remain static, thereby allowing recovering processes to follow the normal unmodified program to reach the time-integration phase. For the time-integration phase, in which the solution data change frequently, remote in-memory checkpointing is used to maintain scalability. This approach is illustrated in Figure 3.

Fenix Fenix is a framework enabling MPI applications to recover nearly transparently from losses of data or computational resources manifested as observable errors. It is based on the premise that the MPI standard itself provides facilities for trapping and isolating such errors, allowing the application to retain control of remaining unaffected resources and obviating full program restart. Building on the success of Fenix's prototype implementation [69, 138], a significant extension has been made to serve many different application needs and styles with *formal specifications* [72]. The current implementation of Fenix leverages ULFM, but the specification does not require it. The ULFM MPI fault tolerance proposal [25] is among the most promising facilities considered for inclusion in the MPI standard.

Low-level application programming interfaces (API) of MPI_ULFM allow a minimal change in the current MPI standard, maintaining backward compatibility. However, a lack of high-level facility, such as application data recovery and rearrangement of MPI ranks after process removal, is deemed too cumbersome to be used by application scientists directly [89]. Fenix's APIs hide the complexity of the low-level MPI operations and provide usable and generic failure recovery patterns for the MPI programming model.

Fenix has two distinct interfaces: process recovery and data recovery. The former allows an MPI application to recover from a permanent loss of MPI processes (ranks) that cause MPI calls to fail. Fenix's data recovery API could be replaced by, or used with, other mechanisms to restore application data.

Fenix's basic assumption is that nominally fatal errors in MPI programs are detected at runtime and reported via error codes. While the default error response is application shutdown (`MPI_ERRORS_ARE_FATAL`), Fenix overrides this, allowing remaining resources (MPI processes) to be informed of the failure, and to call MPI functions to remove the lost resources from their respective communication contexts. To ease adoption of MPI fault tolerance, Fenix automatically captures errors resulting from MPI library calls that experience a failure. Implementations of the Fenix specification can achieve this behaviour

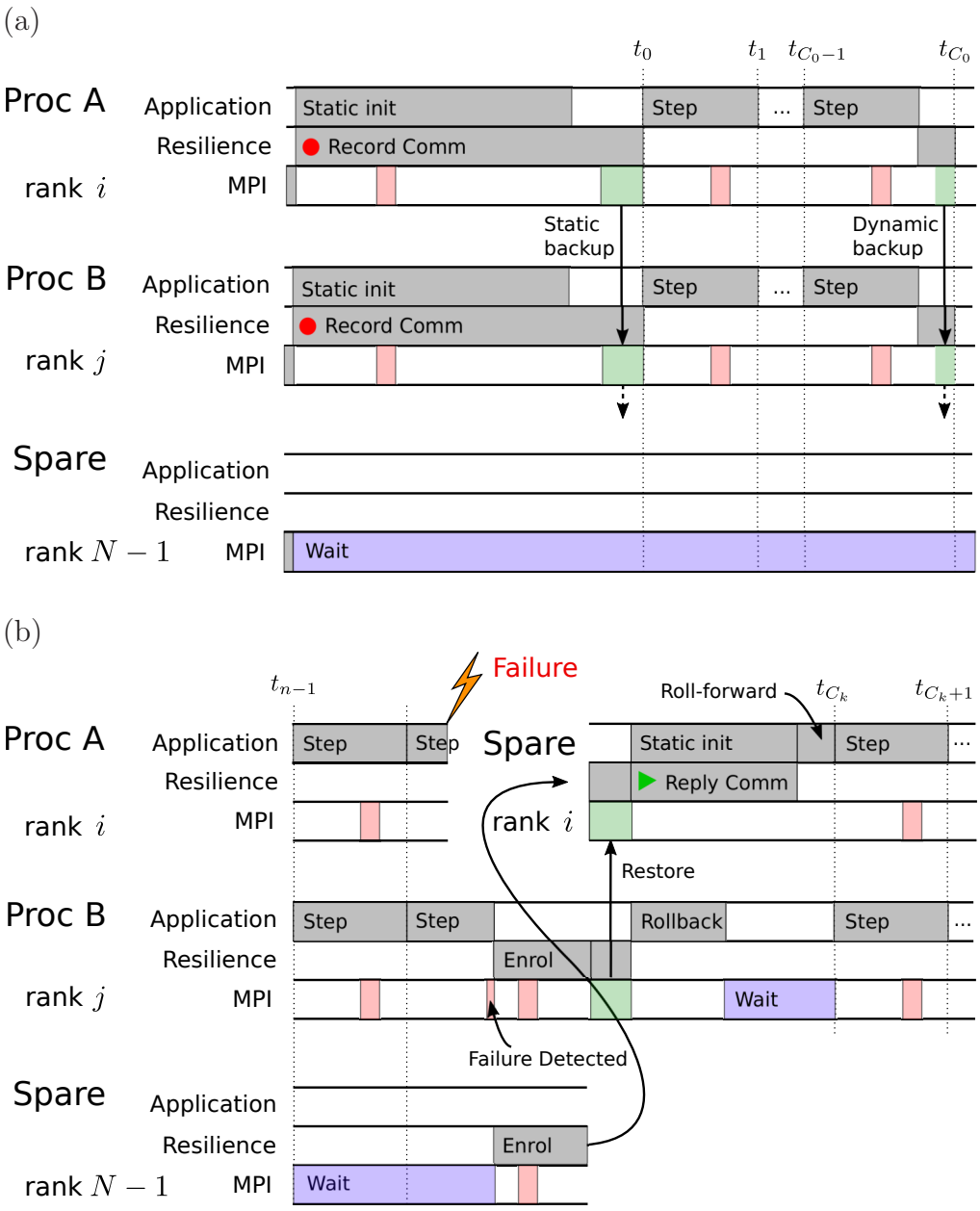


Figure 3: A low intrusive approach to system resilience by splitting the algorithm into a static and dynamic phase [33]. (a) All communication is logged during the static initialisation phase by intercepting MPI function calls. Remote-memory check-pointing is used during the dynamic time-integration phase. (b) Following a failure, spare ranks are used to replace failed ranks. These ranks follow the normal code path but the results of MPI calls are replayed from the message log.

by leveraging error handlers or the MPI profiling interface. Hence, Fenix users need not replace MPI calls with calls to Fenix (for example, `Fenix_Send` instead of `MPI_Send`). Subsequently, Fenix repairs communicators transparently and returns execution control to the application. Its detailed behaviour is determined by function `Fenix_Init`, which also initializes the library.

```
void Fenix_Init(
   MPI_Comm comm,
   MPI_Comm *newcomm,
   int *role,
   int *argc, char ***argv,
   int spare_ranks,
   int spawn,
   MPI_Info info,
   int *error);
```

Above, communicator `comm` is used to create *resilient* communicator `newcomm`. Communicators derived from `newcomm` are automatically resilient. Moreover, errors returned by MPI calls involving resilient communicators are intercepted by Fenix, triggering repair of all resilient communicators, such that the application can resume execution. Control is returned to the application at the logical exit of `Fenix_Init`.

Next, parameter `role` contains the calling rank's most recent history. Possible values are `FENIX_ROLE_`*X*`_RANK`, with *X* being `INITIAL`, `SURVIVOR`, or `RECOVERED`, corresponding to "no errors yet", "not affected by latest failure", and "rank recovered, but has no useful application data", respectively.

Parameter `spare_ranks` specifies how many ranks in `comm` are sequestered in `Fenix_Init` to replace failed ranks, and `spawn` determines whether Fenix may create new ranks (`MPI_Comm_spawn`) to restore resilient communicators to their original size. `spare_ranks` and `spawn` together define Fenix's overall communicator repair policy. If both are zero, resilient communicators are shrunk to exclude failed ranks after an error. If only `spawn` is zero, Fenix draws on the spare ranks pool to restore resilient communicators to their original size. Once the pool is depleted, subsequent errors lead to communicator shrinkage. Parameter `info` conveys details about expected Fenix behaviour that differs from the default.

Once Fenix process recovery has returned an application to a consistent state, the user needs to consider lost data. Sometimes no action is required, for example because the application is embarrassingly parallel, like a Monte Carlo simulation. However, in most

HPC applications, which are Fenix's prime target, ranks synchronize often and the intermediate state of ranks lost due to error must be restored.

We outline several plausible approaches, all supported by Fenix's process and data recovery facilities. However, the user need not use Fenix for data recovery, and may, for example, use Global View Resilience (GVR, [39]) and VeloC [107], a combination of these and Fenix, or others. Fenix chiefly aims at providing fast and in-memory redundant storage for data recovery, whereas GVR and VeloC target secondary storage.

1. *Non-shrinking recovery with full data retrieval.* This is the most common case in bulk-synchronous HPC codes. The programmer defines data/work decomposition that corresponds to a certain number of ranks. After an error, `SURVIVOR`s roll back their state to a prior time. Missing ranks are replaced with `RECOVERED` ranks that instantiate their state using non-local data retrieved by a simple Fenix function invocation. Keeping the same node count is desirable for applications with static data partitioning and bulk-synchronous communication patterns, mitigating the complexity associated with data repartitioning for fewer node counts.
2. *Non-shrinking recovery with local data retrieval only.* `SURVIVOR`s roll back their state, but `RECOVERED` ranks approximate their requisite data, for example by interpolation of logically "nearby" data. This approach, also good for bulk-synchronous codes, may apply to relaxation methods.
3. *Shrinking recovery with full data retrieval.* If the user demands online recovery and resources are insufficient to replace defunct ranks, Fenix shrinks damaged communicators. Now there is no simple, unique way to assign recovered data to the reduced number of remaining ranks. More general, flexible Fenix data recovery functions are provided for alternate ways of retrieving and re-assigning such data.
4. *Shrinking recovery with local data retrieval only.* This is a combination of methods 2 and 3.

To organize redundant storage for data recovery after a fault, Fenix offers *data groups*, containers for sets of data objects (*members*) that are manipulated as a unit. Data groups also refer to the collection of ranks that cooperate in handling recovery data. This collection need not include all active ranks. Fenix adopts the convenient MPI vehicle of *communicators* to indicate the subset of ranks involved.

In addition to the above, Fenix provides query, synchronization, and implicit and explicit garbage collection functions, as well as non-blocking storage functions to improve performance, and functions to manipulate subsets. For more specifications on Fenix we refer to [68, 72, 73].

Kokkos Kokkos [24] is a C++ library designed as a performance-portable node-parallel programming model to allow platform-independent parallel implementations across multiple heterogeneous architectures. The main idea of resilient Kokkos is extending the abstraction of parallel computation (execution space such as `Kokkos::parallel_for`) and data representation (`Kokkos::View` and memory space) to support redundancy. These new features are enabled through template parameters of `Kokkos::parallel_for` and `Kokkos::View` to realize redundant program execution and CPR handling both soft and hard error resilience. During program execution, the resilient Kokkos runtime monitors all active resilient `Kokkos::View` and automatically copies the data inside `Kokkos::View` to the persistent storage as needed. For program recovery, Kokkos exploits the existing I/O and checkpoint library facilitated with annotation capability for `Kokkos::parallel_for` and `Kokkos::View` to locate the point of failure as well as the data objects being lost. Currently, resilient Kokkos provides C++ I/O, HDF5 and VeloC [107] for the persistent storage options and supports OpenMP and CUDA backend for the execution space. Writing library routines and DSL using Kokkos can be done converting array expressions using `Kokkos::View`. The same methodology can be applied to node parallel execution using `Kokkos::parallel_for` and `Kokkos::parallel_reduce`. The reader is referred to [100] for more details on Kokkos resilience capabilities.

Minimal ULFM and fallback This approach is a much more lightweight wrapper around the ULFM specification, specifically targeted at C++ applications that intend to react to node losses and silent data corruption via C++ `exceptions` [60]. The wrapper ensures that in case of a failure an exception can be received on all surviving ranks if it happens inside a guarded block. The term 'surviving' here means, that the rank is capable to continue the computation. We rely on two methods from the ULFM proposal: `MPIX_Comm_revoke`, which revokes the communicator for any communication and `MPIX_Comm_agree` to agree on the error state. Once a rank calls a communication method on a revoked communication an error is raised which is then mapped to an exception. A working

communicator can be recovered by calling `MPIX_Comm_shrink` and its siblings, on which the computation can be continued, after the error state has been resolved. This functionality is typically not available in default MPI installations on clusters yet, and thus, such a wrapper can be convenient in a transition phase.

2.4 Backup grid

To the best of our knowledge, there has been a single study to test an approach to secure weather and climate models against hardware faults on a software level [51]. The approach uses a backup grid to store coarse resolution copies of prognostic variables. In the presence of a hardware fault, an estimate of the original values of the model fields could be reproduced from the backup grid to enable the simulation to continue with no significant delay. The approach could not reproduce a bit-identical result when compared to a fault-free simulation but it could allow the completion of a simulation in the presence of both soft and hard faults.

The backup system uses the following mechanism [51]:

- The prognostic variables from the model grid are mapped onto the coarse-resolution backup grid at the end of each time step. The values of the backup grid are stored for one time step to allow a comparison with the model fields at the following time step.
- It is checked whether the fields on the backup grid have changed by an unexpected amount during a time step. The threshold of this check needs to be tuned to the specific model simulation under consideration. If the change of a model variable is suspicious, it is assumed that a hardware fault has perturbed the simulation. Therefore, the specific value on the backup grid is replaced by the corresponding value from the previous time step.
- The corresponding values on the model grid that influenced the erroneous grid value on the backup grid are checked for values outside of a physically meaningful range.
- If the value on the model grid is found to be unreasonable, it is replaced by the value mapped from the backup grid onto the specific position of the model grid (Figure 4).

The approach was tested on numerical simulations with a two-dimensional shallow water model, a standard test bed for numerical schemes in NWP model development. As long as the backup system was used, simulations did not crash and a high level of solution

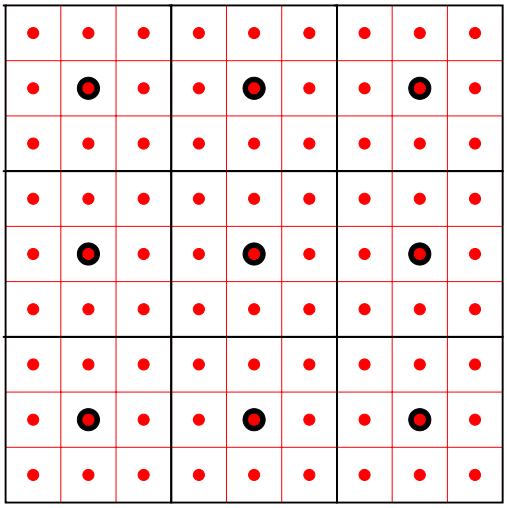


Figure 4: Backup system of [51]. The coarse-resolution backup grid (black grid points) holds an approximated representation of the prognostic variables on the fine-resolution grid (red grid points). If a hardware fault is detected, the prognostic fields on the model grid can be reproduced as an approximation from the backup grid or vice versa.

quality could be maintained. The overhead due to the backup system was reasonable with a 13% runtime increase [51]. Additional storage requirements were small.

However, there are a number of limitations to the approach that would require further research. The backup system, as used in [51], is not able to distinguish between model errors and hardware errors, and it violates local mass conservation and global energy conservation.

2.5 Hardware assistance for fault detection

Reliability, availability and serviceability (RAS) features of an HPC system include all the components required to keep the system functional for long periods without failures. In HPC, it is usually considered that the overhead cost associated with permanent RAS features should be kept small compared to the potential impact of computational nodes loss. This is justified by the balance between cost and performance with respect to application criticality. Indeed, compared to critical applications such as year-round 24/7 real-time safety management systems, it is preferable to restart a simulation job after excluding the failed computational resources, rather than deploying inessential hardware RAS features at high cost. This is es-

pecially true if techniques restarting computations from a previous healthy state exist at application level and can be used on time-consuming or time-limited jobs with minimal impact on the results.

The common hardware RAS features are deployed at different levels, from processors to the whole HPC system. The lower-level RAS features are surveyed in [61], focusing on CPU, memory, intra-node (socket-to-socket) interconnect and emerging FPGA-based hardware accelerators. The authors, from academia, supercomputing centres, and industry, also set a priority for the implementation of each identified low-level resilience feature.

It results in a "must have" features list in production HPC systems based on three main criteria:

1. These resilience features should ensure that the failure rate of the system is below an acceptable threshold depending on the deployed technology, system size and target application;
2. Given the high cost of the uncorrected errors, frequent hardware errors should be corrected, at low cost, preferably (where possible) in hardware;
3. Overheating, one of the main causes of unreliable device behavior, imply that HPC systems should monitor the temperature of their components and include mechanisms that prevent overheating while balancing power/energy and performance.

According to [61], the must have resilience features are:

- At processor and accelerator level:
  - Error detection in CPU caches and registers
  - To avoid overheating
    - Memory thermal throttling: temporarily slowing down the memory access rate to prevent memory modules from overheating.
    - Dynamic voltage and frequency scaling (DVFS) on CPU and accelerator
- At memory level:
  - ECC in main memory and in accelerator memory
  - Memory Demand and Patrol Scrub: feature used in combination with memory error detection (e.g., ECC) to find and correct memory errors, either reactively (demand) or proactively (patrol).
  - Memory Address Parity Protection
    At memory level, ECC is clearly the main feature, also deployed on latest GPU as the Tesla V100 HBM2 memory

subsystem supports Single-Error Correcting Double-Error Detecting ECC to protect data. Yet, ECC can fix only a limited number of errors, so application level techniques are still required.

- At intra-node level:
  - Packet retry in the intra-node interconnect, mostly based on CRC - cyclic redundancy check [112, 116] - checking as done in Intel Ultra Path Interconnect , AMD Infinity Fabric, Cavium Coherent Processor Interconnect.
  - PCIe standard RAS features
- At inter-node level, based on high performance interconnect such as InfiniBand and Ethernet, redundancy is commonly proposed by stacking network switch allowing redundant network links using IEEE 802.3ad Link Aggregation Control Protocol (LACP).
- At storage level, RAID - Redundant Arrays of Inexpensive Disks [111] - is the main common feature deployed commonly in HPC systems in addition with advanced techniques such as Declustered RAID, see [114].

Note that some techniques have a larger positive or negative impact on performance than others. For example, ECC and DVFS features lower the performance while LACP can improve performance as it takes advantage to the redundant hardware.

Redundancy is the approach commonly used at the higher infrastructure level, where management of the redundant resources is especially crucial. At the whole system level, a solution is to have two or more HPC facilities physically apart, as done at ECMWF, Météo France, and the UK Met Office, for example. This method can include also the low-level RAS features.

For concreteness, in the rest of this section we consider in detail a set of high-level hardware redundancy techniques for power, cooling, management and high speed interconnect implemented in a real example, the BullSequana XH2000. This system is relevant for the purposes of NWP applications, since it will be installed at two main weather and climate centres in Europe (ECMWF and Météo France), as well as at supercomputing centres in Europe and beyond, such as GENCI in France, CSC in Finland and C-DAC in India.

In the BullSequana XH2000 there is redundancy at four levels: power, cooling, management and high-speed interconnect. For power redundancy, in the XH2000 design the AC/DC conversion within a rack is shared for all resource elements (compute node, switch, hydraulic chassis, etc.). The power section is composed of

a power distribution unit (PDU), power supply unit (PSU) shelves, optional ultra-capacity module (UCM) and a busbar to distribute power to all the components within the cabinet. The level and type of redundancy is configurable for the PSU shelves, it could be configured at the PSU block or at the PSU shelf level with the following type of configuration: N (no redundancy), $N+1$ redundant, $N+2$ redundant, and $2N$ ($N+N$ redundant). The optional UCM chassis allows the mitigation of micro power outages up to 300ms at full load when 3-phase uninterruptible power supply equipment for system shutdown is not present upstream in the data centre infrastructure. Some HPC centers use an uninterruptible power supply (UPS) so as to maintain the power supply for longer in order to start another power source or properly stop the systems. This method (UPS plus another power source) is so costly with respect to the power at stake that it is rarely deployed. Indeed, the high electrical power demand of current HPC systems is not compatible anymore with the densification and power efficiency imperatives required.

The BullSequana XH2000 Direct Liquid Cooling system is composed of hydraulic chassis (HYC), primary and secondary manifolds, and an expansion tank. The HYCs contain the heat exchanger system that allows it to achieve 95% of heat transfer between the primary and secondary manifolds. Up to 3 HYC are available depending on the redundancy type desired (N or $N+1$). The third HYC is for $N+1$ redundancy only. To protect the system against cooling lost at rack level, the rack is immediately put in low power mode to lower its power consumption, and thus its dissipation, to avoid stopping the production. In case of persistent failure inducing a temperature rise the system shuts down automatically.

The network management of an XH2000 rack is based on a redundant network switch stack (one redundant switch for each switch) and redundant network links using Link Aggregation Control Protocol, ensuring a minimum redundancy. Moreover, in case of a cell failure the cell-based architecture prevents from an impact on the rest of the system.

On the high speed interconnect side, in case of node or switch loss, the Infiniband standard provides for example the possibility to recompute the routing tables in order to find alternative routes and then keep the system running by isolating the failed resources.

3 Example implementations

This section contains illustrative applications of the fault-tolerant algorithmic techniques described in Section 2.

3.1 Resilient GMRES with Interpolation-Restart

The GMRES method is one of the most popular solvers for the solution of non-symmetric linear systems. It belongs to the class of Krylov solvers that minimize the 2-norm of the residual associated with the iterates built in the sequence of Krylov subspaces (MINRES is another example of such a solver [110]). In contrast to many other Krylov methods, GMRES does not update the iterate at each iteration but only either when it has converged or when it restarts every m steps in the so-called restarted GMRES [GMRES(m)] [123].

Most of the parallel GMRES implementations rely on a block row partitioning of the matrix that induces a similar distribution of the Krylov orthonormal basis, while the small upper Hessenberg matrix resulting from the Arnoldi procedure is replicated. The memory footprint and extra local computational cost of this replication are negligible, and it avoids extra expensive global communications.

When a node crashes, the approximate solution is not available. The Hessenberg matrix is replicated on each node and the least squares problem is also solved redundantly. Consequently, each individual node ℓ still in operation can compute its entries I_ℓ of the iterate when a fault occurs. Following the general idea of the Interpolation-Restart policy, the lost entries of the current iterate can be approximated by solving a local linear system defined by Equation (3) or a more robust recovery technique, referred to as LSI (Least Squares Interpolation) that solves the local least squares. If node p fails, $x^{(LSI)}$ is computed as follows:

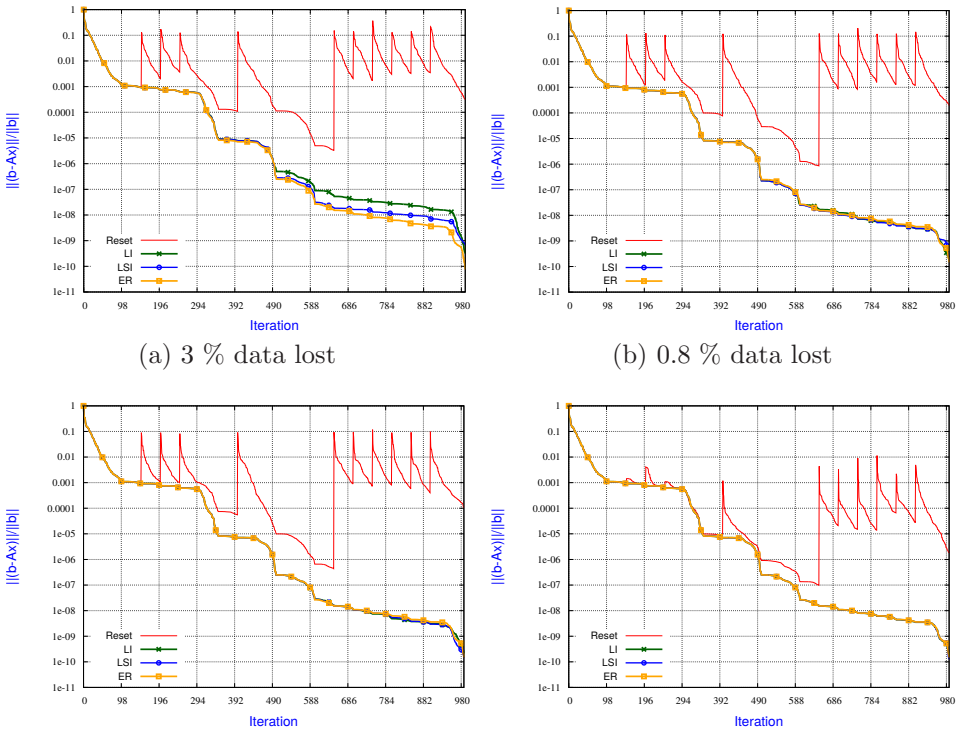
$$\begin{cases} x_{I_p}^{(LSI)} = \underset{x_{I_p}}{\operatorname{argmin}} \| (b - \sum_{q \neq p} A_{:,I_q} x_q^{(k)}) - A_{:,I_p} x_{I_p} \|, \\ x_{I_q}^{(LSI)} = x_{I_q}^{(k)} \qquad \text{for } q \neq p. \end{cases} \tag{4}$$

In addition to removing the assumption on the non-singularity of the A_{I_p,I_p} block made to define the LI policy, the LSI strategy ensures the monotonic decrease of the residual norm, a key property of GMRES.

To study the numerical features of the proposed IR strategies, we display the convergence history as a function of the iterations (Figure 5), that also coincide with the number of preconditioned matrix-vector products.

The recovery policy can induce delay in the convergence for two reasons. The first source of possible delay is the quality of the interpolation, the second is related to the restart that is performed after a fault. To distinguish between the effect of the two sources of delay, we implement a GMRES where we do not inject faults but

impose a restart at each iteration the faulty run experienced a fault (it corresponds to a variable restarted policy for GMRES). We refer to this strategy as Enforced Restart (ER, yellow line in Figure 5). Furthermore, we also depict in red a straightforward strategy where the lost entries of the iterate are replaced by the corresponding ones of the first initial guess. This simple approach is denoted as Reset.



(a) 3 % data lost (b) 0.8 % data lost

(c) 0.2 % data lost (d) 0.001 % data lost

Figure 5: Numerical behaviour of the IR strategies applied to the GMRES(100) solution of the linear system with matrix Averous/epb3 and 10 faults when the amount of data loss is varied.

We solve the linear system $Ax = b$ with GMRES(100) and taking as A the matrix Averous from the Florida matrix collection [42], varying the volume of lost entries between 3% and 0.001% at each fault (a single entry is lost in the latter case). For these experiments, in order to enable cross comparison, the number of faults is kept the same and the faults occur at the same iteration for all the runs. The straightforward restarting Reset policy does not lead to convergence (Figure 5). Each peak in the convergence history corresponds to a fault showing that the solver is not able to converge. These characteristics are similar in other cases like multigrid methods for symmetric positive definite systems [74]. In contrast, the IR strategies ensure convergence and have very similar convergence behaviour and robustness capabilities. Furthermore, the convergence behaviour of IR strategies appears scarcely affected by the amount of data loss.

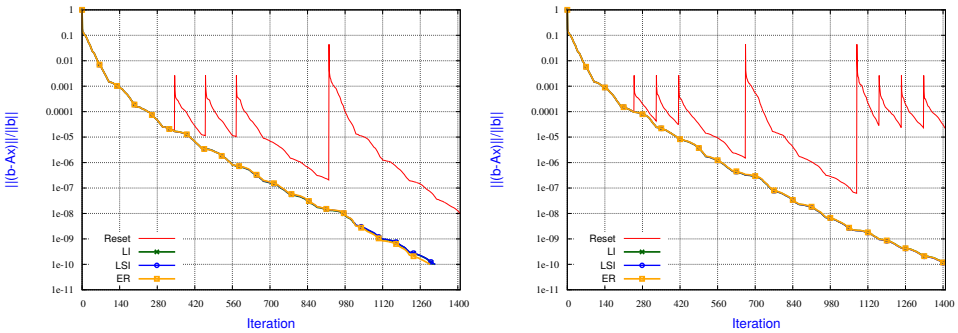
In Figure 6, we investigate the robustness of the IR strategies when the number of faults is varied while the amount of recovered entries remains fixed at 0.2 % after each fault. For those experiments, we consider restarted GMRES(100) to solve a linear system associated with the matrix Kim1 from the Florida matrix collection [42]. An expected general trend that can be observed on this example: the larger the number of faults, the slower the convergence. When only a few faults occur, the convergence penalty is not significant compared to the non-faulty case. For a large number of faults, the convergence is slowed down but continues to take place. For instance, for an expected accuracy of 10^{-7}, the iteration count with 40 faults is twice the one without faults.

3.2 Compressed checkpointing

In order to showcase the backup and recovery options with the compressed checkpointing method, see Section 2 above, we consider the test problem

$$\begin{cases} -\nabla \cdot \left[\begin{pmatrix} 1 & 0 \\ 0 & 0.01 \end{pmatrix} \nabla u \right] = b & \text{in } \Omega = (0,1)^2 \\ u = 0 & \text{on } \partial\Omega \end{cases} \tag{5}$$

with b such that $u(x, y) = \sin(\pi x^2) \sin(\pi y^2)$ is the exact solution. This problem is solved within the DUNE PDE framework by a conjugate gradient method with an algebraic multigrid solver as preconditioner provided by the Iterative Solver Template Library (DUNE-ISTL [27]). The problem is solved in parallel on 52 ranks



(a) 4 faults (b) 8 faults

(c) 17 faults (d) 40 faults

Figure 6: Numerical behaviour of the IR strategies applied to the GMRES(100) solution of the linear system with matrix Kim/kim1, varying number of faults, and fixed 0.2 % data loss at each fault.

using an overlapping Schwarz approach with minimal overlap (essentially using the template `ISTLBackend_CG_AMG_SSOR` as solver from DUNE-PDELab [13]). The iterative solving procedure is stopped after a relative residual reduction of 10^{-8}. In the fault-free case the solver needs 115 iterations to fulfil this criterion for this particular demonstrator problem.

In addition to the recovery approaches, for backup we consider the aforementioned techniques of Zero Backup, Multigrid compression (MG, [74]), and SZ compression, as well as a fourth adaptive SZ compression (aSZ) technique. MG is configured to a fixed backup depth of two which reduces the data volume to 1/16 in our 2D demonstrator. For standard SZ compression, we prescribe a compression accuracy of 1e-7 and 1e-3 as examples for high and low accuracy. For adaptive SZ compression, the compression accuracy is coupled to the normalized local vector norm of the current residual multiplied by an additional tolerance $\texttt{tol}_{\texttt{aSZ}} \in \{$1e-2,1e-1,1e0,1e1$\}$:

$$\frac{\|Ax^{(i)} - b\|_2}{\|b\|_2}\texttt{tol}_{\texttt{aSZ}}. \tag{6}$$

Table 2 shows the average number of iterations to convergence in the iterative solution of problem (5) when a fault happens in iteration $i \in \{10, 40, 75, 110\}$ on rank $r \in \{0, 21, 37\}$, using different combinations of recovery approaches (first three rows) and backup techniques. Furthermore, the last row (aux_iter) shows the number of additional iterations for the local auxiliary problem when the improved recovery is applied, using the different backups as initial guesses.

Local recovery is always superior to global recovery while the overhead is similar or even less. The improved recovery always yields the best possible results but the amount of additional iterations of the auxiliary solver varies. Zero backup, MG and the less accurate SZ compression technique SZ_1e-3 result in a significant number of additional iterations, either global or auxiliary ones. For local recovery, high accurate fixed SZ compression (SZ_1e-7) is good if a fault happens early but is not competitive if a fault happens late in the iterative procedure (e.g. in iteration 110; not visible in the shown Table). Adaptive SZ compression seems to be the best and aSZ_1e0 seems to mark the sweet spot as increasing the accuracy does not yield a big advantage for global and local recovery and a decrease in accuracy (aSZ_1e1) yields around 20% more additional iterations when using improved recovery.

In terms of compression factors (Figure 7, top) and cumulative data size (Figure 7, bottom), most of the SZ compressed backups

Table 2: Average number of iterations to convergence in the solution of the anisotropic diffusion problem (5) with faults, using different approaches for backup and recovery (see text for details).

	Zero	**MG**	**SZ_1e-7**	**SZ_1e-3**	**aSZ_1e-2**	**aSZ_1e-1**	**aSZ_1e0**	**aSZ_1e1**
global	200.00	200.00	154.00	193.00	116.00	115.00	119.00	120.00
local	199.33	194.67	134.33	173.00	114.33	115.67	115.67	114.67
improved	113.00	113.00	113.00	113.00	113.00	113.00	113.00	113.00
aux_iter	190.00	118.33	57.33	106.67	23.00	23.00	24.00	28.33

yield a compression factor of around 100 for the given data size. Therefore, the cumulative size over 115 iterations is less than the size of two classic checkpoints of one iteration. Furthermore, the adaptive approach takes advantage of the low accuracy of the solver in the early stages and obtains higher compression factors. At the end of the iterative solve, the adaptive compression is not as strong as the non-adaptive one, but it enables a more efficient recovery as shown before. It should be noted that SZ compression gives even better compression factors when used for more data points. The cumulative checkpoint size could be further reduced by reducing its frequency, i.e., a checkpoint every n-th iteration. This can also reduce the communication overhead because the backups on different ranks could be created in different iterations and thus the communication would be better distributed.

3.3 Soft fault detection and correction in the Generalised Conjugate Residual method

A new nonhydrostatic dynamical core, the Finite Volume Module (FVM) [88] of the Integrated Forecasting System, is being developed at ECMWF for the next-generation NWP. The module uses a preconditioned Generalised Conjugate Residual (GCR) elliptic solver [130, 132] to solve the boundary value problem that arises from the application of the Non-oscillatory Forward-in-Time (NFT) integrator based on the MPDATA advection scheme [131] and results in an equation which can symbolically be written as

$$\mathcal{L}(\phi) = \mathcal{R}, \tag{7}$$

where $\mathcal{L}$ is the elliptic operator and $\mathcal{R}$ represents the right-hand side of the problem at hand. The elliptic solver (Algorithm 1) belongs to a class of Krylov subspace methods, minimises the L^2-norm of

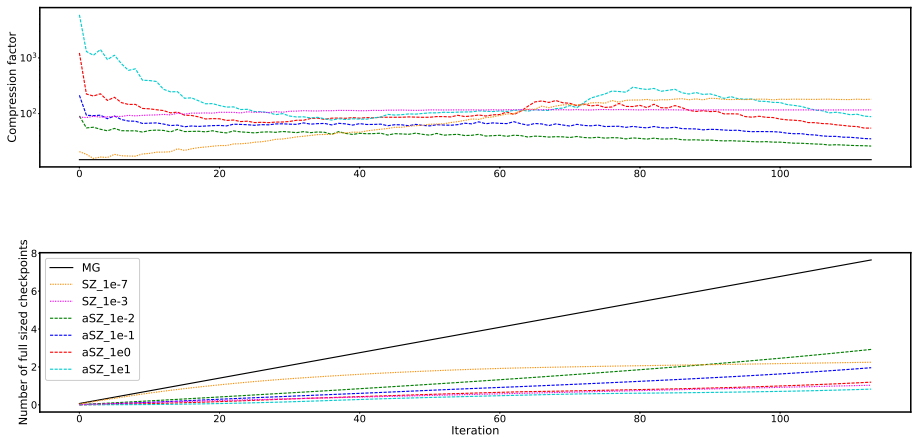


Figure 7: History of the average compression factor (top) and cumulative backup size compared to full-sized checkpoint (bottom) over the iterative process using different backup techniques in the iterative solution of the anisotropic diffusion problem (5) with faults.

the residual, r, of the model (7), and solves the k^{th}-order damped oscillator equation with preconditioner operator $\mathcal{P}$:

$$\frac{\partial^k \mathcal{P}(\phi)}{\partial \tau^k} + \frac{1}{T_{k-1}(\tau)} \frac{\partial^{k-1} \mathcal{P}(\phi)}{\partial \tau^{k-1}} + ... + \frac{1}{T_1(\tau)} \frac{\partial^1 \mathcal{P}(\phi)}{\partial \tau^1} = \mathcal{L}(\phi) - \mathcal{R} \quad (8)$$

See [132] for a description of the physically motivated stopping criteria. The damped oscillator equation (8) is discretised in pseudo-time τ and optimal parameters $T_1, ..T_{k-1}$ are determined to assure the minimisation of $\langle r, r \rangle$ for the field ϕ (see [130, 132] for a detailed discussion).

The solver employs bespoke left-preconditioning (see e.g. [88] for details) to address the large condition number induced by the domain anisotropy for global atmospheric problems. Since the elliptic solver takes a significant ($\sim 40\%$) proportion of the computational resources of the dynamical core, it is quite likely to encounter soft faults if they were to occur during computation.

The iterative nature of GCR provides ample opportunity to recompute faulty data at minimal cost to the solver, given an ability to detect such faults. A basic detection method can be derived, given that for each iteration of GCR (n in Algorithm 1), the k^{th} order dampening takes place on a Krylov subspace, K_n,

Algorithm 1 GCR(k):

For any initial guess, ϕ^0, set $r^0 = \mathcal{L}\left(\phi^0\right) - \mathcal{R}$, $p^0 = \mathcal{P}^{-1}\left(r^0\right)$; then iterate:
for $n = 1, 2, ...$ until convergence **do**
  for $\nu = 0, ..., k-1$ **do**
    $\beta = \dfrac{\langle r^\nu \mathcal{L}\left(p^\nu\right)\rangle}{\langle \mathcal{L}\left(p^\nu\right)\mathcal{L}\left(p^\nu\right)\rangle}$
    $\phi^{\nu+1} = \phi^\nu + \beta p^\nu$
    $r^{\nu+1} = r^\nu + \beta \mathcal{L}\left(p^\nu\right)$
    if $\|r^{\nu+1}\| \leq \epsilon$ **then**
      exit
    end if
    $e = \mathcal{P}^{-1}\left(r^{\nu+1}\right)$
    $\mathcal{L}(e) = \sum_{l=1}^{3} \dfrac{1}{\zeta_l^*} \nabla \cdot \zeta_l^* C \nabla e$
    for $l = 0, ..\nu$ **do**
      $\alpha_l = \dfrac{\left\langle \mathcal{L}(e)\mathcal{L}\left(p^l\right)\right\rangle}{\langle \mathcal{L}\left(p^l\right)\mathcal{L}\left(p^l\right)\rangle}$
      $p^{\nu+1} = e + \sum_{l}^{\nu} \alpha_l p^l$
      $\mathcal{L}\left(p^{\nu+1}\right) = \mathcal{L}(e) + \sum_{l}^{\nu} \alpha_l \mathcal{L}\left(p^l\right)$
    end for
  end for
  reset $[\phi, r, p, \mathcal{L}(p)]^k$ to $[\phi, r, p, \mathcal{L}(p)]^0$
end for

such that $K_n \subseteq K_{n+1}$, $\forall n \in 1, 2, \ldots$. As such, each iteration of GCR minimises the L^2-norm of the residual, r, on that Krylov subspace. Since the norm is non-increasing on each subspace, it is also non-increasing between subspaces. If during computation the discrete L^2-norm value increases, that most likely indicates a problem with the solver. Testing the resilience of Algorithm 1 by injecting faults (bit flips) into various stages of GCR often cause the solver to stall for an iteration, after which the routine usually continues - sometimes more slowly but often without other noticeable signs that a problem has been encountered. Even when a larger proportion of data is corrupted by a fault, GCR is usually able to converge with about $10 - 20\%$ computational overhead. However, this can cause some erratic behaviour of the solver - including large jumps in the maximum absolute value of the residual in the domain, with the potential to induce undesired modes into the solution, even if the solver

itself still converges.

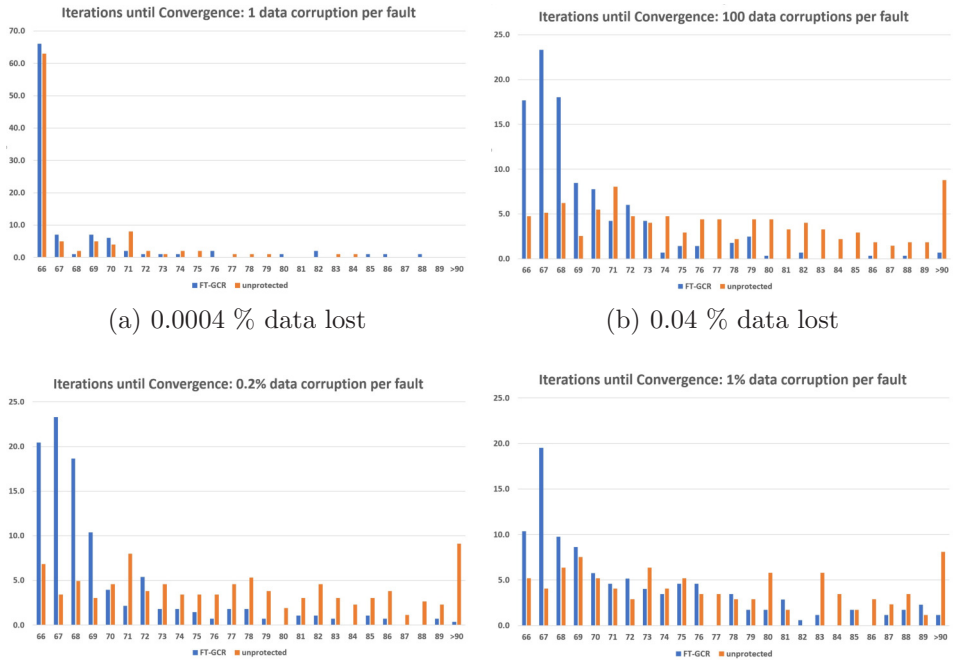
Algorithm 2 FT-GCR(k):

For any initial guess, ϕ^0, set $r^0 = \mathcal{L}\left(\phi^0\right) - \mathcal{R}$, $p^0 = \mathcal{P}^{-1}\left(r^0\right)$; then iterate:
for $n = 1, 2, ...$ until convergence **do**
  for $\nu = 0, ..., k-1$ **do**
    $\beta = \dfrac{\langle r^\nu \mathcal{L}\left(p^\nu\right)\rangle}{\langle \mathcal{L}\left(p^\nu\right)\mathcal{L}\left(p^\nu\right)\rangle}$
    $\phi^{\nu+1} = \phi^\nu + \beta p^\nu$
    $r^{\nu+1} = r^\nu + \beta \mathcal{L}\left(p^\nu\right)$
    if $\|r^{\nu+1}\| \leq \epsilon$ **then**
      exit
    end if
    if $r^{\nu+1} \geq r^\nu$ **then**
      n=n-1
      reset $[\phi, r, p, \mathcal{L}(p)]^0$ to $[\phi, r, p, \mathcal{L}(p)]^*$
    else if $\nu = 0$ **then**
      set $[\phi, r, p, \mathcal{L}(p)]^*$ to $[\phi, r, p, \mathcal{L}(p)]^0$
    end if
    $e = \mathcal{P}^{-1}\left(r^{\nu+1}\right)$
    $\mathcal{L}(e) = \sum_{l=1}^{3} \dfrac{1}{\zeta_l^*} \nabla \cdot \zeta_l^* C \nabla e$
    for $l = 0, ..\nu$ **do**
      $\alpha_l = \dfrac{\left\langle \mathcal{L}(e)\mathcal{L}\left(p^l\right)\right\rangle}{\langle \mathcal{L}\left(p^l\right)\mathcal{L}\left(p^l\right)\rangle}$
      $p^{\nu+1} = e + \sum_l^\nu \alpha_l p^l$
      $\mathcal{L}\left(p^{\nu+1}\right) = \mathcal{L}(e) + \sum_l^\nu \alpha_l \mathcal{L}\left(p^l\right)$
    end for
  end for
  reset $[\phi, r, p, \mathcal{L}(p)]^k$ to $[\phi, r, p, \mathcal{L}(p)]^0$
end for

Assuming a fault has been detected, the solver can be easily reverted to a state previously deemed good (i.e., no fault detected), at the cost of at most a full iteration of algorithm 1 (if a fault occurs during the $\nu = k-1$ pass). Algorithm 2 describes the fault tolerant GCR (FT-GCR) method, including snap-shotting and fault detection.

In order to test the effectiveness of FT-GCR (algorithm 2), the GCR elliptic solver in [97] was adapted to include fault injection, detection, and resetting. The method solves potential flow for an isolated hill on the planet. An O32 Octahedral

reduced Gaussian mesh (corresponding to $\sim$ 280km horizontal resolution) with 51 vertical layers was used for the simulations.



(a) 0.0004 % data lost

(b) 0.04 % data lost

(c) 0.2 % data lost

(d) 1 % data lost

Figure 8: Numerical behaviour of GCR and FT-GCR when faults are introduced. Each histogram displays the percentage of runs that converged at a given iteration. Fault free convergence is reached after 66 iterations.

Faults are injected after the preconditioning stage with an implausibly high 2% probability, with a maximum of 10 individual fault occurrences allowed. The fault causes a given number of entries of $e = P^{-1}\left(r^{\nu+1}\right)$, the preconditioner output, to suffer a bit flip. Figure 8 illustrates graphically the response of the protected and unprotected simulations, for different levels of data corruption per fault encountered. Histograms a), b), c), and d) correspond to 1 (0.0004%), 100 (0.04%), 500 (0.2%), and 2700 (1%) entries corrupted (% of total data) respectively. Table 3 documents averaged statistics for each of these simulation sets. Each set contains between 100 and 200 simulation results, where faultless simulations are disregarded.

It is clear that GCR is highly resilient to bit flip faults, as even when a larger amount of data is corrupted the unprotected algorithm usually manages to converge. Additionally, it can be

Table 3: The average number of faults per run, average number of faults detected, average iterations for FT-GCR, and the average number of iterations for unprotected GCR for each plot in Figure 8.

Histogram	% data lost	Faults per run	Faults detected	Convergence FT-GCR	Convergence GCR
a	0.0004%	4.17	0.1	68.0	68.0
b	0.04%	4.38	3.72	69.4	77.1
c	0.2%	4.33	3.66	69.6	77.8
d	1.0%	4.61	3.28	72.2	77.3

observed that minor data corruption events have little effect on convergence (Figure 8a), which is also the likely reason why such small corruption events are difficult to detect. When a sufficient proportion of data is corrupted, FT-GCR detects between $60 - 80\%$ of faults (Table 3), and convergence delay is reduced by 5 to 8 iterations compared to the unprotected runs. The overhead of this fault tolerance method is small, with one backup step required per full GCR iteration - including a small amount of extra memory to store the backups. The cost of detection is also minimal, since the expensive global sum to find the L^2-norm of the residual is already computed.

4 Discussion

The techniques surveyed in this paper to secure NWP applications against faults are not yet implemented within operational environments, but may start to play a significant role in the next few years.

As exascale architectures start to be deployed, the feasibility of disk or memory CPR will ultimately be contingent on the evolution of memory latency and bandwidth compared with the growing size of restart files. In addition, to assume that all calculations will be performed with no faults will become more costly. Computational cost could be significantly reduced if constraints for fault-free calculations could be weakened (see for example discussion of ECC memory checking and stochastic

processors in Section 1.3).

The ongoing transition to exascale NWP models constitutes an excellent framework to set out objectives and requirements for resilience and fault-tolerance as well as to test possible options to include resilience features in future atmospheric models. More economical resilience strategies should be investigated that could flank or supplant CPR, which should ideally become the technique of last resort. On the road to those solutions, decisions and compromises need to be made on a number of aspects, for instance whether to focus just on fault recovery or also fault detection, and on hard or soft faults.

In the United States, efforts are under way in the framework of the Exascale computing project in order to improve hardware resilience. In particular, application of VeloC [108] has enabled to significantly reduce CPR time [48]. With a final project target of a one-week MTBF at exascale, on given machines a three-fold increase has been obtained.

In Europe, at ECMWF energy efficiency and model sustainability efforts are focusing on reshaping model implementation into more manageable independent units called dwarfs, developed in the framework of the ESCAPE [104] and ESCAPE-2 projects. The dwarfs are eventually expected to cover around 60 per cent of the forecast model code, so improved checkpointing procedures can be tried on existing dwarfs with a view to establishing a resilience framework applying to future ones. For instance, experiments with checkpointing in memory [33] rather than to disk could be carried out. Avoiding the parallel filesystem may conceivably give a sizeable boost in performance at large scale. With this and other possible solutions, performance gains from fault recovery using spares and spawning, as well as savings in requeue time and start-up cost of NWP code should be weighed against development overhead for the tools themselves. From the systems point of view, standards are already starting to provide resilience tools, an example being the inclusion of ULFM into version 4 of MPI as an extension.

In addition, as part of efforts to ensure code portability and multi-architecture performance, NWP models are transitioning to implementations based on domain-specific-language (DSL) approaches, such as Kokkos [24], Psyclone [1], and Stella/GridTools [67, 139]. Fault-tolerance efforts requiring code modifications will perforce interface with DSLs, and related investigations

have already started, for which see Section 2.3 above.

On the hardware side, securing stronger support from HPC vendors in resilience efforts will require clear use cases from the NWP community. ESCAPE and ESCAPE-2 dwarfs can provide an optimal environment for such experiments. When a dwarf-tested system resilience technique evolves into a de facto standard in the weather and climate community, it can become a differentiator for vendors to address this HPC market.

While vendors have generally been reluctant to publish hardware resilience statistics such as MTBF for HPC architectures, such information could be included in future procurements. Hardware could include system monitoring functionalities, so that users can isolate nodes that look suspicious and distinguish between healthy and unhealthy ones. In fact, unlike faults on data, soft faults impacting instruction sets of hardware as well as integer arithmetic are not addressed yet by any of the methods in this survey, which have not yet been tested for a specific hardware configuration. Atos Bull, an ESCAPE and ESCAPE-2 project member, takes part in MPI Forum, the MPI standardization body [103]. Currently there is a strong interest in the high-performance Fault Tolerance Interface (FTI) for hybrid systems [16], and Atos Bull is already delivering FTI as part of its software stack called Super Computer Suite.

From the algorithms viewpoint, properties of the methods shown in this paper could be explored in more general NWP settings. In particular, interpolation-restart techniques could be deployed to fault-proof linear solvers in atmospheric dynamical cores under development such as the ECMWF's FVM [88] or the Discontinuous Galerkin (DG) dynamical core under development in ESCAPE-2 along the lines proposed in [140].

As shown in the resilience experiments with an FT-GCR solver presented in this paper, a small numbers of faults in the elliptic solver are both hard to detect and generally have reduced impact on convergence. If we can better detect soft faults in the elliptic solver, the proposed repair technique would be able to adequately protect this part of the model. FVM uses a GCR solver, so that implementing the FT-GCR version introduced here would give an immediate comparison between it and, for example, IR-GMRES in terms of resilience and overhead. Results could be compared to those obtained with disk checkpointing or replication to evaluate trade-offs and inform

decisions.

Next, compressed and in-memory checkpointing as seen in Section 2.2 could be incorporated into the FT-GCR solver. Assuming that checkpointing will have to play a more dominant role to maintain robustness of solvers, our experiments show that compression can increase efficiency and reduce overhead substantially, without compromising the numerical accuracy. Using these techniques in an adaptive way yields low overhead at the start of the iterative procedure, when a restart or information loss has little effect. Later, when a simple restart will cause the loss of a lot of invested resources and progress, the compression accuracy increases as well as the overhead but the recovery is as good as with no compression in return. The computational overhead introduced by the SZ compression is in the low single-digit percentage range while its achieved compression factors can decrease the memory overhead significantly. Small scale numerical tests (cf. Figure 7) already provide promising compression factors although SZ compression requires much larger data sets to exploit its full potential. Introducing these techniques into the FT-GCR solver could further reduce its communication and storage overhead while maintaining its recovery properties. In addition, backup grid approaches as seen in Section 2.4 can easily be implemented in other numerical frameworks, for example in the modal DG framework by storage of the lowest order modal coefficients of the solution for each element.

Finally, it can be argued that the operational use of ensemble simulations for weather and climate introduces another level of resilience. Since individual ensemble members are run in parallel, it may not be efficient to scale a single simulation to the size of the entire supercomputer. If a single simulation is crashing due to a hardware fault, at least most of the members may finish in time and still allow the delivery of a forecast. However, it will be difficult to perform reliable forecasts if the number and quality of ensemble simulations varies. In general, the error of ensemble predictions will be proportional to the sum of a constant that is dependent on the quality of the forecast model plus a term which is proportional to the inverse of the number of ensemble members.

5 Conclusion

Performance of HPC facilities is expected to grow by an order of magnitude and cross the exascale threshold in the next decade. Numerical weather and climate prediction models are at the forefront for exploiting the available computational power and are preparing to tackle the challenges to their systems, from data assimilation all the way to product dissemination. Projected figures for failure frequencies question the assumptions on fault-free operating cycles that scientists have been taking for granted after the early days of numerical computing. As research teams and operational centres overhaul their codes to rise up to the exascale challenge, enhanced scalable performance will be of little use on algorithms operating with reduced precision, plagued by unchecked soft faults, and within workflows based on unaffordable checkpointing schedules running on machines with unknown reliability.

This report explored pathways to algorithmic fault-tolerance and computational resilience adequate for models used in numerical weather and climate prediction. Existing numerical techniques were surveyed, ranging from interpolation-restart and compressed checkpointing approaches for iterative solvers to in-memory checkpointing, ULFM-based process and data recovery for MPI, and advanced backup techniques. While this set of algorithms and approaches was deemed a fair picture of the state of the art, other methods are available and the field is under rapid development. The methods were chosen because of their proximity to NWP practice - iterative solvers for linear systems being a major component of semi-implicit methods employed in most operational dynamical cores worldwide, MPI being the de facto standard for CPU-based distributed-memory parallelism.

Compressed checkpointing aside, the methods analyzed here chiefly, though not exclusively, concern recovery from hard faults. The frequency of soft errors in modern supercomputers presently does not appear to be very precisely known to computational scientists and practitioners. Bit-reproducibility tests could be relatively easily set up for large model configurations in order to start filling this knowledge gap. Hardware vendors could offer valuable advice in these efforts, which would have to be intertwined with verification and validation (V&V) strategies.

Indeed, bit-reproducibility tests are a common verification step in the testing of NWP models, but the protocol for these tests and the interpretation of their results may need to be revisited if more frequent soft faults have to be taken into account. In addition to soft failures, the use of massively threaded systems (accelerators), FPGA, and special-purpose – such as AI – chips creates a huge challenge for bitwise reproducibility. Overall bit-reproducibility is realistic when executing only a few test runs, while model checking can be used in workflows with repeated code execution. For the latter, it is essential to explore more sophisticated V&V techniques to assess the correctness of NWP code. Today, a variety of V&V techniques have been practiced in other areas of computational science and engineering [120, 121], and the high performance computing community have been recognizing the importance of V&V techniques in the areas of compiler optimization [21], runtime (MPI and thread scheduler [36, 62]) and numerical kernels [46].

In order to strengthen the case for hardware and software resilience, NWP scientists should therefore reach out to other scientific communities confronted with related issues. In particular, any nonlinear dynamic model will face similar challenges to NWP in terms of soft error spread. Given the breadth and depth of applications, machine learning efforts currently exert strong impact on hardware developments. In view of the data sizes in play there, and of the growing use of those techniques next to, or instead of, standard approaches in numerical weather prediction, an investigation on the need for resilience in machine learning could offer valuable insights and help to drive hardware development.

Wide-ranging multinational exascale projects currently under way provide ideal scientific arenas in which to discuss and develop mitigating strategies against the threats posed by increasing fault rates. The pressing need for reliable and time-critical production of weather and climate forecasts makes a compelling case for accelerating the pace of such endeavours.

Acknowledgments

The present paper stems from the ESCAPE-2 workshop on fault tolerant algorithms and resilient approaches for exascale computing held in January 2019 at the Mathematics Department of Politecnico di Milano. An earlier version of the present paper was submitted as an internal ESCAPE-2 project deliverable. We thank the authors of [4, 6], namely E. Agullo, L. Giraud, A. Guermouche, J. Roman, P. Salas, and M. Zounon, for the permission to report the description and numerical testing of the interpolation-restart strategy in Sections 2.1 and 3. This work was supported by the ESCAPE-2 project, European Union's Horizon 2020 research and innovation programme (grant agreement No 800897); the ESiWACE2 Centre of Excellence,European Union's Horizon 2020 research and innovation programme (grant agreement No 823988); and the Deutsche Forschungsgemeinschaft under Germany's Excellence Strategy – EXC-2075 – (grant agreement no. 390740016). PD gratefully acknowledges funding from the Royal Society for his University Research Fellowship.

Note

References

[1] S.V. Adams, R.W. Ford, M. Hambley, J.M. Hobson, I. Kavčič, C.M. Maynard, T. Melvin, E.H. Müller, S. Mullerworth, A.R. Porter, M. Rezny, B.J. Shipway, and R. Wong. LFRic: Meeting the challenges of scalability and performance portability in weather and climate models. *Journal of Parallel and Distributed Computing*, 132:383–396, 2019.

[2] E. Agullo, S. Cools, L. Giraud, A. Moreau, P. Salas, W. Vanroose, E. F. Yetkin, and M. Zounon. Hard faults and soft-errors: Possible numerical remedies in linear algebra solvers. In *International Conference on Vector and Parallel Processing*, pages 11–18. Springer, 2016.

[3] E. Agullo, L. Giraud, A. Guermouche, J. Roman, and M. Zounon. Towards resilient parallel linear Krylov solvers: recover-restart strategies. Technical report, INRIA, 2013.

[4] E. Agullo, L. Giraud, A. Guermouche, J. Roman, and M. Zounon. Numerical recovery strategies for parallel resilient Krylov linear solvers. *Numerical Linear Algebra with Applications*, 23(5):888–905, 2016.

[5] E. Agullo, L. Giraud, and Y.-F. Jing. Block GMRES method with inexact breakdowns and deflated restarting. *SIAM Journal on Matrix Analysis and Applications*, 35(4):1625–1651, 2014.

[6] E. Agullo, L. Giraud, P. Salas, and M. Zounon. Interpolation-restart strategies for resilient eigensolvers. *SIAM Journal on Scientific Computing*, 38(5):C560–C583, 2016.

[7] E. Agullo, L. Giraud, and M. Zounon. On the resilience of parallel sparse hybrid solvers. In *2015 IEEE 22nd International Conference on High Performance Computing (HiPC)*, pages 75–84. IEEE, 2015.

[8] M. M. Ali, J. Southern, P. Strazdins, and B. Harding. Application level fault recovery: Using Fault-Tolerant Open MPI in a PDE solver. In *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pages 1169–1178. IEEE, 2014.

[9] M. Altenbernd and D. Göddeke. Soft fault detection and correction for multigrid. *The International Journal of High Performance Computing Applications*, 32(6):897–912, 2018.

[10] R. A. Ashraf, S. Hukerikar, and C. Engelmann. Shrink or substitute: Handling process failures in HPC systems using in-situ recovery. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 178–185. IEEE, 2018.

[11] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33, 2004.

[12] A.D. Barker, D.E. Bernholdt, A.S. Bland, J.D. Gary, J.J. Hack, S.T. McNally, J.H. Rogers, B. Smith, T.P. Straatsma, S.R. Sukumar, et al. High Performance Computing Facility Operational Assessment 2015 Oak Ridge Leadership Computing Facility. *ORNL*, 2016.

[13] P. Bastian, F. Heimann, and S. Marnach. Generic implementation of finite element methods in the Distributed and Unified Numerics Environment (DUNE). *Kybernetika*, 46:294–315, 2010.

[14] P. Bauer, A. Thorpe, and G. Brunet. The quiet revolution of numerical weather prediction. *Nature*, 525(7567):47–55, 2015.

[15] L. Bautista-Gomez and F. Cappello. Detecting silent data corruption for extreme-scale MPI applications. In *Proceedings of the 22nd European MPI Users' Group Meeting*, page 12. ACM, 2015.

[16] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and S. Matsuoka. FTI: high performance fault tolerance interface for hybrid systems. In *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2011.

[17] L. Bautista-Gomez, F. Zyulkyarov, O. Unsal, and S. McIntosh-Smith. Unprotected computing: A large-scale study of dram raw error rate on a supercomputer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 55. IEEE Press, 2016.

[18] T. Benacchio. *A blended semi-implicit numerical model for weakly compressible atmospheric dynamics*. PhD thesis, Freie Universität Berlin, 2014. [Available at: `http://edocs.fu-berlin.de/diss/receive/FUDISS\_thesis\_000000097149`].

[19] T. Benacchio, W. O'Neill, and R. Klein. A blended soundproof-to-compressible model for atmospheric dynamics. *Monthy Weather Review*, 142(12):4416–4438, 2014.

[20] A. Benoit, A. Cavelan, F. Cappello, P. Raghavan, Y. Robert, and H. Sun. Identifying the right replication level to detect and correct silent errors at scale. In *Proceedings of the 2017 Workshop on Fault-Tolerance for HPC at Extreme Scale*, pages 31–38. ACM, 2017.

[21] M. Bentley, I. Briggs, G. Gopalakrishnan, D. H. Ahn, I. Laguna, Gregory L. Lee, and H. E. Jones. Multi-level analysis of compiler-induced variability and performance tradeoffs. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, page 61–72. ACM, 2019.

[22] 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 the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 275–278. ACM, 2015.

[23] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, and F. Cappello. Exploring partial replication to improve lightweight silent data cor-

ruption detection for HPC applications. In *European Conference on Parallel Processing*, pages 419–430. Springer, 2016.

[24] L. Bertagna, M. Deakin, O. Guba, D. Sunderland, A.M. Bradley, I.K. Tezaur, M. A. Taylor, and A.G. Salinger. HOMMEXX 1.0: a performance-portable atmospheric dynamical core for the Energy Exascale Earth System Model. *Geoscientific Model Development*, 12(4):1423–1441, 2019.

[25] W. Bland, A. Bouteiller, T. Herault, Gorge Bosilca, and Jack J. Dongarra. Post-failure recovery of MPI communication capability: Design and rationale. *International Journal of High Performance Computing Applications*, 2013.

[26] W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bosilca, and J. J. Dongarra. An evaluation of user-level failure mitigation support in MPI. *Computing*, 95(12):1171–1184, 2013.

[27] M. Blatt and P. Bastian. The Iterative Solver Template Library. In Bo Kågström, Erik Elmroth, Jack Dongarra, and Jerzy Waśniewski, editors, *Applied Parallel Computing. State of the Art in Scientific Computing*, pages 666–675. Springer Berlin Heidelberg, 2007.

[28] M. Blatt, A. Burchardt, A. Dedner, Ch. Engwer, J. Fahlke, B. Flemisch, Ch. Gersbacher, C. Gräser, F. Gruber, Ch. Grüninger, D. Kempf, R. Klöfkorn, T. Malkmus, S. Müthing, M. Nolte, M. Piatkowski, and O. Sander. The Distributed and Unified Numerics Environment, Version 2.4. *Archive of Numerical Software*, 4(100):13–29, 2016.

[29] R. Bonaventura, L. Redler and R. Budich. *Earth System Modelling 2: Algorithms, Code Infrastructure and Optimisation.* Springer Verlag, New York, 2012.

[30] A. Bouras and V. Frayssé. Inexact matrix-vector products in Krylov methods for solving linear systems: a relaxation strategy. *SIAM Journal on Matrix Analysis and Applications*, 26(3):660–678, 2005.

[31] P. G. Bridges, K. B. Ferreira, M. A. Heroux, and M. Hoemmen. Fault-tolerant linear solvers via selective reliability. *arXiv preprint arXiv:1206.1390*, 2012.

[32] J. Calhoun, M. Snir, L. N. Olson, and W. D. Gropp. Towards a more complete understanding of SDC propagation. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, pages 131–142. ACM, 2017.

[33] C. D. Cantwell and A. S. Nielsen. A minimally intrusive low-memory approach to resilience for existing transient solvers. *Journal of Scientific Computing*, 78(1):565–581, 2019.

[34] F. Cappello. Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities. *The International Journal of High Performance Computing Applications*, 23(3):212–226, 2009.

[35] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir. Toward exascale resilience: 2014 update. *Supercomputing frontiers and innovations*, 1(1):5–28, 2014.

[36] D. Chapp, T. Johnston, and M. Taufer. On the need for reproducible numerical accuracy through intelligent runtime selection of reduction algorithms at the extreme scale. In *2015 IEEE International Conference on Cluster Computing*, pages 166–175, 2015.

[37] C. Chen, Y. Du, K. Zuo, J. Fang, and C. Yang. Toward fault-tolerant hybrid programming over large-scale heterogeneous clusters via checkpointing/restart optimization. *The Journal of Supercomputing*, pages 1–22, 2017.

[38] C.-Y. Cher, M. Gupta, P. Bose, and K. P. Muller. Understanding soft error resiliency of blue gene/Q compute chip through hardware proton irradiation and software fault injection. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 587–596. IEEE, 2014.

[39] A. Chien, P. Balaji, P. Beckman, N. Dun, A. Fang, H. Fujita, K. Iskra, Z. Rubenstein, Z. Zheng, R. Schreiber, et al. Versioned distributed arrays for resilience in scientific applications: Global view resilience. *Procedia Computer Science*, 51:29–38, 2015.

[40] I. Cores, G. Rodríguez, P. González, R. R. Osorio, et al. Improving scalability of application-level checkpoint-recovery by reducing checkpoint sizes. *New Generation Computing*, 31(3):163–185, 2013.

[41] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future generation computer systems*, 22(3):303–312, 2006.

[42] T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *j-TOMS*, 38(1):1:1–1:25, 2011.

[43] A. Dawson, P. D. Düben, D. A. MacLeod, and T. N. Palmer. Reliable low precision simulations in land surface models. *Climate Dynamics*, 51(7):2657–2666, 2018.

[44] D. De Oliveira, L. Pilla, M. Hanzich, V. Fratin, F. Fernandes, C. Lunardi, J. M. Cela, P. Navaux, L. Carro, and P. Rech. Radiation-induced error criticality in modern HPC parallel accelerators. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 577–588. IEEE, 2017.

[45] T. J. Dell. A white paper on the benefits of chipkill-correct ECC for PC server main memory. *IBM Microelectronics Division*, 11:1–23, 1997.

[46] J. Demmel and H. D. Nguyen. Parallel reproducible summation. *IEEE Transactions on Computers*, 64(7):2060–2070, 2015.

[47] S. Di and F. Cappello. Fast error-bounded lossy HPC data compression with SZ. In *2016 IEEE international parallel and distributed processing symposium*, pages 730–739. IEEE, 2016.

[48] C. Di Martino, W. Kramer, Z. Kalbarczyk, and R. Iyer. Measuring and understanding extreme-scale application resilience: A field study of 5,000,000 HPC application runs. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 25–36. IEEE, 2015.

[49] J. Dongarra, T. Herault, and Y. Robert. Fault tolerance techniques for high-performance computing. In *Fault-Tolerance Techniques for High-Performance Computing*, pages 3–85. Springer, 2015.

[50] P. Du, A. Bouteiller, G. Bosilca, T. Herault, and J. Dongarra. Algorithm-based fault tolerance for dense matrix factorizations. *ACM SIGPLAN notices*, 47(8):225–234, 2012.

[51] P. D. Düben and A. Dawson. An approach to secure weather and climate models against hardware faults. *Journal of Advances in Modeling Earth Systems*, 9(1):501–513, 2017.

[52] P. D. Düben and S. I. Dolaptchiev. Rounding errors may be beneficial for simulations of atmospheric flow: results from the forced 1D Burgers equation. *Theoretical and Computational Fluid Dynamics*, 29:311–328, 2015.

[53] P. D. Düben, J. Joven, A. Lingamneni, H. McNamara, G. De Micheli, K. V. Palem, and T. N. Palmer. On the use of inexact, pruned hardware in atmospheric modelling. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 372(2018):20130276, 2014.

[54] P. D. Düben, H. McNamara, and T.N. Palmer. The use of imprecise processing to improve accuracy in weather & climate prediction. *Journal of Computational Physics*, 271:2–18, 2014.

[55] P. D. Düben and T. N. Palmer. Benchmark tests for numerical weather forecasts on inexact hardware. *Monthly Weather Review*, 142(10):3809–3829, 2014.

[56] P. D. Düben, N. P. Wedi, C. Zeman, and S. Saarinen. Global simulations of the atmosphere at 1.45 km grid-spacing with the Integrated Forecasting System. *Journal of the Meteorological Society of Japan. Ser. II*, 2020. `https://doi.org/10.2151/jmsj.2020-016`.

[57] J. Elliott, M. Hoemmen, and F. Mueller. Evaluating the impact of SDC on the GMRES iterative solver. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 1193–1202. IEEE, 2014.

[58] J. Elliott, M. Hoemmen, and F. Mueller. A numerical soft fault model for iterative linear solvers. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 271–274. ACM, 2015.

[59] J. Elliott, M. Hoemmen, and F. Mueller. Exploiting data representation for fault tolerance. *Journal of computational science*, 14:51–60, 2016.

[60] C. Engwer, M. Altenbernd, N.-A. Dreier, and D. Göddeke. A high-level C++ approach to manage local errors, asynchrony and faults in an MPI application. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 714–721. IEEE, 2018.

[61] European HPC resilience initiative. Towards Resilient EU HPC Systems: A Blueprint. Whitepaper, Forthcoming, 2020.

[62] Noah Evans. Verifying Qthreads: Is model checking viable for user level tasking runtimes? In I. Laguna and C. Rubio-González, editors, *2nd IEEE/ACM International Workshop on Software Correctness for HPC Applications, CORRECTNESS@SC 2018, Dallas, TX, USA, November 12, 2018*, pages 25–32. IEEE, 2018.

[63] G. E. Fagg and J. J. Dongarra. FT-MPI: Fault tolerant MPI, supporting dynamic applications in a dynamic world. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pages 346–353. Springer, Berlin, Heidelberg, 2000.

[64] S. Feng, S. Gupta, A. Ansari, and S. Mahlke. Shoestring: probabilistic soft error reliability on the cheap. In *Proceedings of ASPLOS XV*, pages 385–396. ACM, 2010.

[65] 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*, page 78. IEEE Computer Society Press, 2012.

[66] O. Fuhrer, T. Chadha, T. Hoefler, G. Kwasniewski, X. Lapillonne, D. Leutwyler, D. Lüthi, C. Osuna, C. Schär, T. C. Schulthess, and H. Vogt. Near-global climate simulation at 1 km resolution: establishing a performance baseline on 4888 GPUs with COSMO 5.0. *Geoscientific Model Development*, 11(4):1665–1681, 2018.

[67] O. Fuhrer, C. Osuna, X. Lapillonne, T. Gysi, B. Cumming, M. Bianco, A. Arteaga, and T. C. Schulthess. Towards a performance portable, architecture agnostic implementation strategy for weather and climate models. *Supercomputing frontiers and innovations*, 1(1):45–62, 2014.

[68] M. Gamell, D. S. Katz, K. Teranishi, M. A. Heroux, R. F. Van der Wijngaart, T. G. Mattson, and M. Parashar. Evaluating online global recovery with Fenix using application-aware in-memory checkpointing techniques. In *2016 45th International Conference on Parallel Processing Workshops (ICPPW)*, pages 346–355. IEEE, 2016.

[69] M. Gamell, D.S. Katz, H. Kolla, J. Chen, S. Klasky, and M. Parashar. Exploring automatic, online failure recovery for scientific applications at extreme scales. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 895–906. IEEE, 2014.

[70] M. Gamell, K. Teranishi, H. Kolla, J. Mayo, M. A Heroux, J. Chen, and M. Parashar. Scalable Failure Masking for Stencil Computations

using Ghost Region Expansion and Cell to Rank Remapping. *SIAM Journal on Scientific Computing*, 39(5):S347–S378, 2017.

[71] M. Gamell, K. Teranishi, J. Mayo, H. Kolla, M. A. Heroux, J. Chen, and M. Parashar. Modeling and simulating multiple failure masking enabled by local recovery for stencil-based applications at extreme scales. *IEEE Transactions on Parallel and Distributed Systems*, 28(10):2881–2895, 2017.

[72] M. Gamell, K. Teranishi, R. Van Der Wijngaart, E. Valenzuela, M. Heroux, and M. Parashar. Fenix, a fault tolerant programming framework for MPI applications 1.1. Technical Report SAND No. 2016-xxxxx, Sandia National Laboratories, Livermore, CA, 2016.

[73] M. Gammel, R. Van Der Wijngaart, K. Teranishi, and M. Parashar. Specification of Fenix MPI Fault Tolerance library version 1.0. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2016.

[74] D. Göddeke, M. Altenbernd, and D. Ribbrock. Fault-tolerant finite-element multigrid algorithms with hierarchically compressed asynchronous checkpointing. *Parallel Computing*, 49:117–135, 2015.

[75] Rachid Guerraoui and André Schiper. Software-based replication for fault tolerance. *Computer*, 30(4):68–74, 1997.

[76] P.-L. Guhur, E. Constantinescu, D. Ghosh, T. Peterka, and F. Cappello. Detection of silent data corruption in adaptive numerical integration solvers. In *Cluster Computing (CLUSTER), 2017 IEEE International Conference on*, pages 592–602. IEEE, 2017.

[77] S. Gupta, T. Patel, C. Engelmann, and D. Tiwari. Failures in large scale systems: long-term measurement, analysis, and implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 44. ACM, 2017.

[78] A. Hassani, A. Skjellum, and R. Brightwell. Design and evaluation of FA-MPI, a transactional resilience scheme for non-blocking MPI. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 750–755. IEEE, 2014.

[79] S. Hatfield, P. Düben, M. Chantry, K. Kondo, T. Miyoshi, and T. Palmer. Choosing the optimal numerical precision for data assimilation in the presence of model error. *Journal of Advances in Modeling Earth Systems*, 10(9):2177–2191, 2018.

[80] M. F. Hoemmen, M. A. Heroux, K. B. Ferreira, and P. G. Bridges. Fault-tolerant iterative methods via selective reliability. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2011.

[81] K.-H. Huang and J.A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE transactions on computers*, 100(6):518–528, 1984.

[82] M. Huber, B. Gmeiner, U. Rüde, and B. Wohlmuth. Resilience for massively parallel multigrid solvers. *SIAM Journal on Scientific Computing*, 38(5):S217–S239, 2016.

[83] T. Z. Islam, K. Mohror, S. Bagchi, A. Moody, B.R. De Supinski, and R. Eigenmann. McrEngine: a scalable checkpointing system using data-aware aggregation and compression. *Scientific Programming*, 21(3-4):149–163, 2013.

[84] S. Kannan, A. Gavrilovska, K. Schwan, and D. Milojicic. Optimizing checkpoints using NVM as virtual memory. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 29–40, 2013.

[85] J. Kim, M. Sullivan, and M. Erez. Bamboo ECC: Strong, safe, and flexible codes for reliable computer memory. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 101–112, 2015.

[86] J. Kim, M. Sullivan, S.-L. Gong, and M. Erez. Frugal ECC: Efficient and versatile memory error protection through fine-grained compression. In *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 1–12. ACM, 2015.

[87] N. Kohl, J. Hötzer, F. Schornbaum, M. Bauer, C. Godenschwager, H. Köstler, B. Nestler, and U. Rüde. A scalable and extensible checkpointing scheme for massively parallel simulations. *The International Journal of High Performance Computing Applications*, page 1094342018767736, 2017.

[88] C. Kühnlein, W. Deconinck, R. Klein, S. Malardel, Z. P. Piotrowski, P. K. Smolarkiewicz, J. Szmelter, and N. P. Wedi. FVM 1.0: A nonhydrostatic finite-volume dynamical core formulation for IFS. *Geoscientific Model Development Discussions*, 12:651–676, 2019.

[89] I. Laguna, D. F. Richards, T. Gamblin, M. Schulz, and B.R. de Supinski. Evaluating user-level fault tolerance for MPI applications. In *Proceedings of the 21st European MPI Users' Group Meeting*, page 57. ACM, 2014.

[90] J. Langou, Z. Chen, G. Bosilca, and J. Dongarra. Recovery patterns for iterative methods in a parallel unstable environment. *SIAM Journal on Scientific Computing*, 30:102–116, 2007.

[91] D. Li, Z. Chen, P. Wu, and J. S. Vetter. Rethinking algorithm-based fault tolerance with a cooperative software-hardware approach. In *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2013.

[92] G. Li, K. Pattabiraman, S.K.S. Hari, M. Sullivan, and T. Tsai. Modeling soft-error propagation in programs. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 27–38. IEEE, 2018.

[93] X. Liang, S. Di, D. Tao, Si. Li, Sh. Li, H. Guo, Z. Chen, and F. Cappello. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. *2018 IEEE International Conference on Big Data (Big Data)*, pages 438–447, 2018.

[94] A. Lingamneni, K. K. Muntimadugu, C. Enz, R. M. Karp, K. V. Palem, and C. Piguet. Algorithmic methodologies for ultra-efficient inexact architectures for sustaining technology scaling. In *Proceedings of the 9th Conference on Computing Frontiers*, pages 3–12. ACM, 2012.

[95] N. Losada, G. Bosilca, A. Bouteiller, P. González, and M.J. Martín. Local rollback for resilient MPI applications with application-level checkpointing and message logging. *Future Generation Computer Systems*, 91:450–464, 2019.

[96] R.E. Lyons and W. Vanderkulk. The use of triple-modular redundancy to improve computer reliability. *IBM journal of research and development*, 6(2):200–209, 1962.

[97] G. Mengaldo. Batch 1: Definition of several weather & climate dwarfs. *Tech. rep., ECMWF*, 2016. `https://arxiv.org/abs/1908.06089`.

[98] H. Meuer, E. Strohmaier, J. J. Dongarra, and H. D. Simon. Top500 Supercomputer Sites. `http://www.top500.org/`, November 2019.

[99] S. E. Michalak, A. J. DuBois, C. B. Storlie, H. M. Quinn, W. N. Rust, D. H. DuBois, D. G. Modl, A. Manuzzato, and S. P. Blanchard. Assessment of the impact of cosmic-ray-induced neutrons on hardware in the Roadrunner supercomputer. *IEEE Transactions on Device and Materials Reliability*, 12(2):445–454, 2012.

[100] J. S. Miles, K. Teranishi, N. M. Morales, and C. R. Trott. Software resilience using Kokkos ecosystem. Technical Report SAND2019-3616, Sandia National Laboratories, 2019.

[101] S. Mittal and J. S. Vetter. A survey of software techniques for using non-volatile memories for storage and main memory systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(5):1537–1550, 2016.

[102] S. Mittal and J.S. Vetter. A survey of techniques for modeling and improving reliability of computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(4):1226–1238, 2015.

[103] MPIForum. `https://www.mpi-forum.org`, 2020. Last accessed: 2019-11-22.

[104] A. Müller, W. Deconinck, C. Kühnlein, G. Mengaldo, M. Lange, N. Wedi, P. Bauer, P. K. Smolarkiewicz, M. Diamantakis, S.-J. Lock, M. Hamrud, S. Saarinen, G. Mozdzynski, D. Thiemert, M. Glinton, P. Bénard, F. Voitus, C. Colavolpe, P. Marguinaud, Y. Zheng, J. Van Bever, D. Degrauwe, G. Smet, P. Termonia, K. P. Nielsen, B. H. Sass, J. W. Poulsen, P. Berg, C. Osuna, O. Fuhrer, V. Clement, M. Baldauf, M. Gillard, J. Szmelter, E. O'Brien, A. McKinstry, O. Robinson, P. Shukla, M. Lysaght, M. Kulczewski, M. Ciznicki, W. Piatek, S. Ciesielski, M. Błazewicz, K. Kurowski, M. Procyk, P. Spychala, B. Bosak, Z. P. Piotrowski, A. Wyszogrodzki, E. Raffin, C. Mazauric, D. Guibert, L. Douriez, X. Vigouroux, A. Gray,

P. Messmer, A. J. Macfaden, and N. New. The ESCAPE project: Energy-efficient Scalable Algorithms for Weather Prediction at Exascale. *Geoscientific Model Development*, 12(10):4425–4441, 2019.

[105] P. Neumann, P. Düben, P. Adamidis, P. Bauer, M. Brück, L. Kornblueh, D. Klocke, B. Stevens, N. Wedi, and J. Biercamp. Assessing the scales in numerical weather and climate predictions: will exascale be the rescue? *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 377(2142):20180148, 2019.

[106] X. Ni, E. Meneses, N. Jain, and L.V. Kalé. ACR: Automatic checkpoint/restart for soft and hard error protection. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 7. ACM, 2013.

[107] B. Nicolae, A. Moody, E. Gonsiorowski, K. Mohror, and F. Cappello. VeloC: Towards high performance adaptive asynchronous checkpointing at large scale. `https://veloc.readthedocs.io/en/latest/`, 2019.

[108] B. Nicolae, A. Moody, E. Gonsiorowski, K. Mohror, and F. Cappello. VeloC: Towards High Performance Adaptive Asynchronous Checkpointing at Large Scale. In *IPDPS'19: The 2019 IEEE International Parallel and Distributed Processing Symposium*, pages 911–920, Rio de Janeiro, Brazil, 2019.

[109] A. S. Nielsen. *Scaling and Resilience in Numerical Algorithms for Exascale Computing*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2018. Available at `infoscience.epfl.ch/record/258087/files/EPFL_TH8926.pdf`.

[110] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12:617–629, 1975.

[111] D.A. Patterson and J.L. Hennessy. Parallelism and the memory hierarchy: Redundant arrays of inexpensive disks. In *Computer Organization and Design ARM Edition*, chapter 5.11. Morgan Kaufmann, Cambridge, MA, USA, 2016.

[112] W. W. Peterson and D. T. Brown. Cyclic codes for error detection. *Proceedings of the IRE*, 49(1):228–235, 1961.

[113] J. S. Plank, M. Beck, and G. Kingsley. Compiler-assisted memory exclusion for fast checkpointing. *IEEE Technical Committee on Operating Systems and Application Environments*, 7(4):10–14, 1995.

[114] Z. Qiao, S. Liang, S. Fu, H. Chen, and B. Settlemyer. Characterizing and modeling reliability of declustered raid for hpc storage systems. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks – Industry Track*, pages 17–20, 2019.

[115] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical mathematics*, volume 37. Springer Science & Business Media, 2010.

[116] T. V. Ramabadran and S. S Gaitonde. A tutorial on CRC computations. *IEEE micro*, 8(4):62–75, 1988.

[117] F. Rizzi, K. Morris, K. Sargsyan, P. Mycek, C. Safta, O. Le Maître, O. Knio, and B. Debusschere. Partial differential equations preconditioner resilient to soft and hard faults. *The International Journal of High Performance Computing Applications*, 32(5):658–673, 2018.

[118] F. Rizzi, K. Morris, K. Sargsyan, P. Mycek, C. Safta, O. Le Maître, O. Knio, and B. Debusschere. Exploring the interplay of resilience and energy consumption for a task-based partial differential equations preconditioner. *Parallel Computing*, 73:16–27, 2018.

[119] G. Rodríguez, M. J. Martín, P. González, J. Tourino, and R. Doallo. CPPC: a compiler-assisted tool for portable checkpointing of message-passing applications. *Concurrency and Computation: Practice and Experience*, 22(6):749–766, 2010.

[120] C. J. Roy. Review of code and solution verification procedures for computational simulation. *Journal of Computational Physics*, (205):131–156, 2005.

[121] C. J. Roy and W. L. Oberkampf. A comprehensive framework for verification, validation, and uncertainty quantification in scientific computing. *Computer Methods in Applied Mechanics and Engineering*, 200(25):2131 – 2144, 2011.

[122] F. P. Russell, P. D. Düben, X. Niu, W. Luk, and T. N. Palmer. Architectures and precision analysis for modelling atmospheric variables with chaotic behaviour. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 171–178, 2015.

[123] Y. Saad and M. H. Schultz. GMRES: A Generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[124] J.C. Sancho, F. Petrini, G. Johnson, and E. Frachtenberg. On the feasibility of incremental checkpointing for scientific computing. In *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, page 58. IEEE, 2004.

[125] P. Sao and R. Vuduc. Self-stabilizing iterative solvers. In *Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, page 4. ACM, 2013.

[126] K. Sargsyan, F. Rizzi, P. Mycek, C. Safta, K. Morris, H. Najm, O. Le Maître, O. Knio, and B. Debusschere. Fault resilient domain decomposition preconditioner for PDEs. *SIAM Journal on Scientific Computing*, 37(5):A2317–A2345, 2015.

[127] J. Sartori, J. Sloan, and R. Kumar. Stochastic computing: embracing errors in architecture and design of processors and applications. In *Proceedings of the 14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, pages 135–144. ACM, 2011.

[128] B. Schroeder and G. Gibson. A large-scale study of failures in high-performance computing systems. *IEEE transactions on Dependable and Secure Computing*, 7(4):337–350, 2009.

[129] T. C. Schulthess, P. Bauer, N. Wedi, O. Fuhrer, T. Hoefler, and C. Schär. Reflecting on the goal and baseline for Exascale Computing: A roadmap based on weather and climate simulations. *Computing in Science & Engineering*, 21(1):30–41, 2018.

[130] P. K. Smolarkiewicz and L. Margolin. Variational methods for elliptic problems in fluid models. *In:Proc. Workshop on Developments in Numerical Methods for Very High Resolution Global Models, 5-7 June 2000, ECMWF, Reading, UK, 137-159*, pages 137–159, 2000.

[131] P. K. Smolarkiewicz and J. Szmelter. MPDATA: An edge-based unstructured-grid formulation. *Journal of Computational Physics*, 206:624–649, 2005.

[132] P. K. Smolarkiewicz and J. Szmelter. A nonhydrostatic unstructured-mesh soundproof model for simulation of internal gravity waves. *Acta Geophysica*, 59:1109–1134, 2011.

[133] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, et al. Addressing failures in exascale computing. *The International Journal of High Performance Computing Applications*, 28(2):129–173, 2014.

[134] J. Steppeler, R. Hess, G. Doms, U. Schättler, and L. Bonaventura. Review of numerical methods for nonhydrostatic weather prediction models. *Meteorology and Atmospheric Physics*, 82:287–301, 2003.

[135] G. Sun. *Exploring Memory Hierarchy Design with Emerging Memory Technologies*. Springer, 2014.

[136] D. Tao, S. Di, Z. Chen, and F. Cappello. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1129–1139. IEEE, 2017.

[137] D. Tao, S. Di, X. Liang, Z. Chen, and F. Cappello. Improving performance of iterative methods by lossy checkponting. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, pages 52–65. ACM, 2018.

[138] K. Teranishi and M. A. Heroux. Toward local failure local recovery resilience model using MPI-ULFM. In *Proceedings of the 21st European MPI Users' Group Meeting*, page 51. ACM, 2014.

[139] F. Thaler, S. Moosbrugger, C. Osuna, M. Bianco, H. Vogt, A. Afanasyev, L. Mosimann, O. Fuhrer, T. C. Schulthess, and T. Hoefler. Porting the COSMO Weather Model to Manycore CPUs. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, page 13. ACM, 2019.

[140] G. Tumolo and L. Bonaventura. A semi-implicit, semi-Lagrangian discontinuous Galerkin framework for adaptive numerical weather

prediction. *Quarterly Journal of the Royal Meteorological Society*, 141(692):2582–2601, 2015.

[141] D. Turnbull and N. Alldrin. Failure prediction in hardware systems. Technical report, University of California, San Diego, 2003.

[142] A.A. White. A view of the equations of meteorological dynamics and various approximations. *Large-scale atmosphere–ocean dynamics*, 1:1–100, 2002.

MOX Technical Reports, last issues

Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

20/2020 Almi, S.; Belz, S.; Micheletti, S.; Perotto, S.
A DIMENSION-REDUCTION MODEL FOR BRITTLE FRACTURES ON THIN SHELLS WITH MESH ADAPTIVITY

19/2020 Stella, S.; Vergara, C.; Maines, M.; Catanzariti, D.; Africa, P.; Demattè, C.; Centonze, M.; Nob
Integration of maps of activation times in computational cardiac electrophysiology

16/2020 Paolucci, R.; Mazzieri, I.; Piunno, G.; Smerzini, C.; Vanini, M.; Ozcebe, A.G.
Earthquake ground motion modelling of induced seismicity in the Groningen gas field

17/2020 Cerroni, D.; Formaggia, L.; Scotti, A.
A control problem approach to Coulomb's friction

18/2020 Fumagalli, A.; Scotti, A.; Formaggia, L.
Performances of the mixed virtual element method on complex grids for underground flow

15/2020 Fumagalli, I.; Fedele, M.; Vergara, C.; Dede', L.; Ippolito, S.; Nicolò, F.; Antona, C.; Scrofani,
An Image-based Computational Hemodynamics Study of the Systolic Anterior Motion of the Mitral Valve

13/2020 Pozzi S.; Domanin M.; Forzenigo L.; Votta E.; Zunino P.; Redaelli A.; Vergara C.
A data-driven surrogate model for fluid-structure interaction in carotid arteries with plaque

14/2020 Calissano, A.; Feragen, A; Vantini, S.
Populations of Unlabeled Networks: Graph Space Geometry and Geodesic Principal Components

11/2020 Antonietti, P.F.; Facciola', C.; Houston, P.; Mazzieri, I.; Pennes, G.; Verani, M.
High-order discontinuous Galerkin methods on polyhedral grids for geophysical applications: seismic wave propagation and fractured reservoir simulations

10/2020 Bonaventura, L.; Carlini, E.; Calzola, E.; Ferretti, R.
Second order fully semi-Lagrangian discretizations of advection--diffusion--reaction systems