

1 Resiliency in Numerical Algorithm Design for 2 Extreme Scale Simulations

3 **Emmanuel Agullo**
Inria

4 **Hartwig Anzt**
KIT – Karlsruher Institut für Technologie

5 **Tommaso Benacchio**
Politecnico di Milano

6 **Hans-Joachim Bungartz**
TU München

7 **Florina M. Ciorba**
Universität Basel

8 **Daniel Drzisga**
TU München

9 **Christian Engelmann**
Oak Ridge National Laboratory

10 **Luc Giraud**
Inria

11 **Marco Heisig**
Universität Erlangen-Nürnberg

12 **Nils Kohl**
Universität Erlangen-Nürnberg

13 **Romain Lion**
University of Bordeaux

14 **Paul Mycek**
Cerfacs

15 **Enrique S. Quintana-Ortí**
Universitat Politècnica de València

16 **Ulrich Rüde**
Universität Erlangen-Nürnberg

17 **Fred Fung**
Australian National University

18 **Linda Stals**
Australian National University

19 **Samuel Thibault**
University of Bordeaux

20 **Andreas Wagner**
21 TU München

Mirco Altenbernd
Universität Stuttgart

Leonardo Bautista-Gomez
Barcelona Supercomputing Center

Luca Bonaventura
Politecnico di Milano

Sanjay Chatterjee
NVIDIA Corporation

Nathan DeBardeleben
Los Alamos National Laboratory

Sebastian Eibl
Universität Erlangen-Nürnberg

Wilfried N. Gansterer
University of Vienna

Dominik Göddeke
Universität Stuttgart

Fabienne Jézéquel
Université Paris 2 – Paris

Xiaoye Sherry Li
Lawrence Berkeley National Laboratory

Miriam Mehl
Universität Stuttgart

Michael Obersteiner
TU München

Francesco Rizzi
NexGen Analytics

Martin Schulz
TU München

Robert Speck
Jülich Supercomputing Centre

Keita Teranishi
Sandia National Laboratories – California

Dominik Thönnies
Universität Erlangen-Nürnberg

Barbara Wohlmuth
TU München



© Luc Giraud, Ulrich Rüde and Linda Stals;
licensed under Creative Commons License CC-BY

Resiliency in Numerical Algorithm Design for Extreme Scale Simulations.

Editors: John Q. Open and Joan R. Access; Article No. ; pp.:1–:57

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

22 **Acronyms**

- 23 **ABFT** *Algorithm-Based Fault Tolerance.* 22, 35
24 **AMR** *Adaptive Mesh Refinement.* 25, 26, 32
25 **API** *Application Programming Interface.* 11, 15, 16

26 **BLCR** *Berkeley Lab Checkpoint/Restart.* 9

27 **CDs** *Containment Domains.* 13, 14
28 **CPU** *Central Processing Unit.* 11
29 **CRC** *Cyclic Redundancy Checks.* 9

30 **DLS** *Dynamic Loop Self-scheduling.* 16
31 **DLS4LB** *Dynamic Loop Scheduling for Load Balancing.* 14
32 **DMR** *Double modular redundancy.* 20
33 **DRAM** *Dynamic Random-Access Memory.* 13
34 **DSL** *Domain Specific Languages.* 38
35 **DUE** *Detectable, but Uncorrectable Error.* 31

36 **ECC** *Error Correcting Codes.* 9, 11, 31

37 **FEM** *Finite Element Method.* 34
38 **FFT** *Fast Fourier Transforms.* 18
39 **FPGA** *Field-Programmable Gate Arrays.* 14

40 **GPU** *Graphics Processing Units.* 11, 14, 32
41 **GVR** *Global View Resilience.* 14

42 **HBM** *High-Bandwidth Memory.* 13
43 **HPC** *High Performance Computing.* 6, 7, 9–13, 15–17, 31, 32, 39

44 **LFLR** *Local-Failure Local-Recovery.* 21, 26

45 **MDS** *Meta Data Service.* 11
46 **MPI** *Message Passing Interface.* 9

47 **NVM** *Non-Volatile Memory.* 13

48 **ODE** *ordinary differential equations.* 19, 20, 35
49 **OS** *Operating Systems.* 9, 32

50 **PCG** *Preconditioned Conjugate Gradient.* 19, 22
51 **PDE** *Partial Differential Equations.* 19–21, 24, 34, 35
52 **PFS** *Parallel File System.* 8, 11, 13
53 **PMPI** *MPI Profiling Interface.* 11
54 **PVFS** *Parallel Virtual File System.* 11

55 **QOS** *Quality of Service.* 11

56 **rDLB** *robust Dynamic Load Balancing.* 15, 16

57 **SDC** *Silent Data Corruption.* 8, 10, 12
58 **SSD** *Solid-State Drives.* 13

59 **TMR** *Triple Modular Redundancy.* 20

60 **ULFM** *User Level Failure Mitigation.* 13, 15

Abstract

This work is based on the seminar titled “Resiliency in Numerical Algorithm Design for Extreme Scale Simulations” held March 1-6, 2020 at Schloss Dagstuhl, that was attended by all the authors. Advanced supercomputing is characterized by very high computation speeds at the cost of involving an enormous amount of resources and costs. A typical large-scale computation running for 48 hours on a system consuming 20 MW, as predicted for exascale systems, would consume a million kWh, corresponding to about 100k Euro in energy cost for executing 10^{23} floating-point operations. It is clearly unacceptable to lose the whole computation if any of the several million parallel processes fails during the execution. Moreover, if a single operation suffers from a bit-flip error, should the whole computation be declared invalid? What about the notion of reproducibility itself: should this core paradigm of science be revised and refined for results that are obtained by large scale simulation?

Naive versions of conventional resilience techniques will not scale to the exascale regime: with a main memory footprint of tens of Petabytes, synchronously writing checkpoint data all the way to background storage at frequent intervals will create intolerable overheads in runtime and energy consumption. Forecasts show that the mean time between failures could be lower than the time to recover from such a checkpoint, so that large calculations at scale might not make any progress if robust alternatives are not investigated.

More advanced resilience techniques must be devised. The key may lie in exploiting both advanced system features as well as specific application knowledge. Research will face two essential questions: (1) what are the reliability requirements for a particular computation and (2) how do we best design the algorithms and software to meet these requirements? While the analysis of use cases can help understand the particular reliability requirements, the construction of remedies is currently wide open. One avenue would be to refine and improve on system- or application-level checkpointing and rollback strategies in the case an error is detected. Developers might use fault notification interfaces and flexible runtime systems to respond to node failures in an application-dependent fashion. Novel numerical algorithms or more stochastic computational approaches may be required to meet accuracy requirements in the face of undetectable soft errors. These ideas constituted an essential topic of the seminar.

The goal of this Dagstuhl Seminar was to bring together a diverse group of scientists with expertise in exascale computing to discuss novel ways to make applications resilient against detected and undetected faults. In particular, participants explored the role that algorithms and applications play in the holistic approach needed to tackle this challenge. This article gathers a broad range of perspectives on the role of algorithms, applications, and systems in achieving resilience for extreme scale simulations. The ultimate goal is to spark novel ideas and encourage the development of concrete solutions for achieving such resilience holistically.

This article gathers a broad range of perspectives on the role of algorithms, applications, and systems in achieving resilience for extreme scale simulations. The ultimate goal is to spark novel ideas and encourage the development of concrete solutions for achieving such resilience holistically.

2012 ACM Subject Classification Theory of computation → Massively parallel algorithms; Networks → Error detection and error correction; Computing methodologies → Parallel programming languages; Computer systems organization → Dependable and fault-tolerant systems and networks

Keywords and phrases Numerical algorithms, Parallel computer architecture, Fault tolerance, Resilience

Digital Object Identifier 10.4230/LIPIcs.Seminar.20101.2020.

106 **Contents**

107	1 Introduction	6
108	2 System infrastructure techniques for resilience	8
109	2.1 Detected and transparently corrected errors	9
110	2.2 Detected errors mitigated with assistance	12
111	2.2.1 Correction with incremental redesign	12
112	2.2.2 Correction with major redesign	15
113	3 Numerical algorithms for resilience	17
114	3.1 Error detecting algorithms	17
115	3.1.1 Exceptions	17
116	3.1.2 Checksums	17
117	3.1.3 Constraints	19
118	3.1.4 Technical error information	19
119	3.1.5 Multi-resolution	20
120	3.1.6 Redundancy	20
121	3.2 Error aware algorithms	21
122	3.2.1 Error aware algorithms for the solution of linear systems	21
123	3.2.2 Error aware algorithms for the solution of partial differential equations	24
124	3.3 Error oblivious algorithms	28
125	3.3.1 Gossip based methods	28
126	3.3.2 Fixed-point methods	28
127	3.3.3 Krylov subspace solvers	29
128	3.3.4 Domain decomposition	30
129	3.3.5 Time stepping	30
130	4 Future directions	30
131	4.1 Systems in support of resilient algorithms	31
132	4.1.1 Error correcting codes	31
133	4.1.2 Improving checkpoint/restart	31
134	4.1.3 Scheduler and resource management	32
135	4.2 Programming models with inherent resiliency support	33
136	4.3 Future directions for the solution of partial differential equations	33
137	4.3.1 Redundancy and replication	33
138	4.3.2 Hierarchy and mixed precision	34
139	4.3.3 Error control	35
140	4.3.4 Locality, asynchronicity and embarassingly parallelism	35
141	4.3.5 Stochastic	36
142	4.3.6 Iterative methods	36
143	4.3.7 Low memory footprint – matrix-free	37
144	4.4 The final mile: towards a resilient ecosystem	37
145	4.4.1 Tools to support resilience software development	37
146	4.4.2 User/Programmer education	39
147	5 Conclusions	39

148 **1 Introduction**

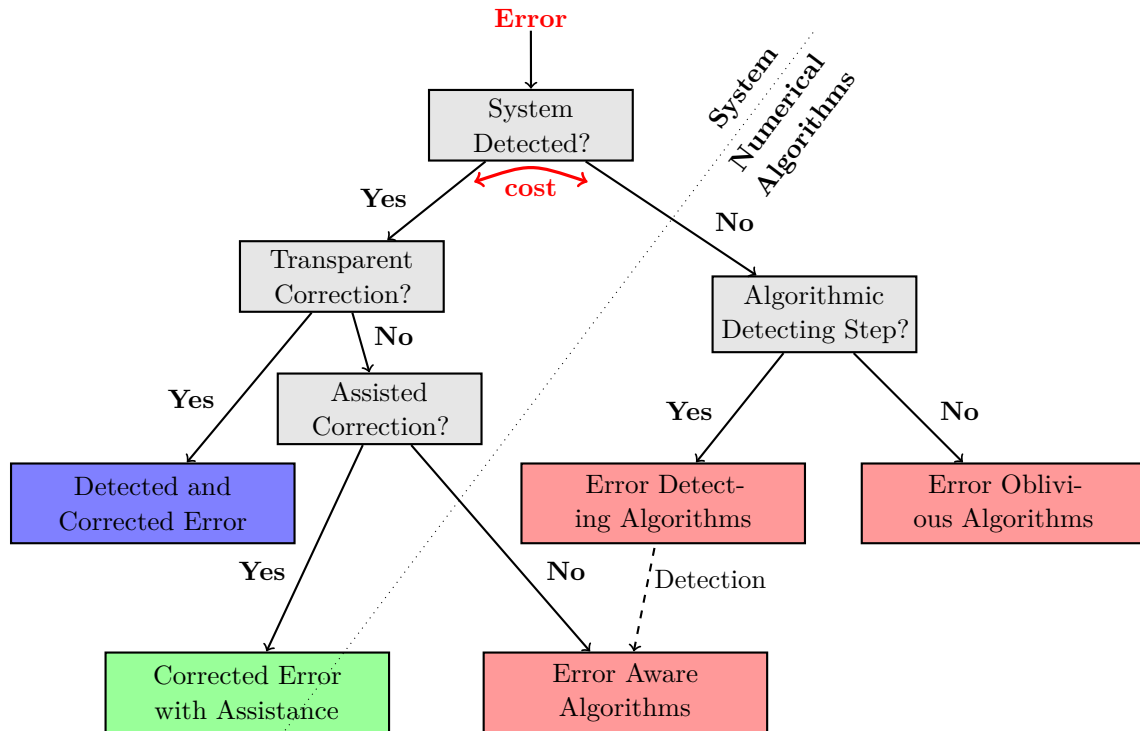
149 Numerical simulation is the third pillar in science discovery at the same level as theory
 150 and experiments. To cope with the ever demanding computational resources needed by
 151 complex simulations, the computational power of high performance computing systems
 152 continues to increase by using an ever larger number of cores or by specialized processing.
 153 On the technological side, the continuous shrinking of transistor geometry and the increasing
 154 complexity of these devices affect their sensitivity to external effects and thus diminish their
 155 reliability. A direct consequence is that *High Performance Computing* (HPC) applications
 156 are increasingly prone to errors. Therefore the design of resilient systems and numerical
 157 algorithms that are able to exploit possible unstable HPC platforms has become a major
 158 concern in the computational science community. To tackle this critical challenge on the path
 159 to extreme scale computation an holistic and multidisciplinary approach is required that needs
 160 to involve researchers from various scientific communities ranging from the hardware/system
 161 community to applied mathematics for the design of novel numerical algorithms. In this
 162 article, we summarize and report on the outcomes of a Dagstuhl seminar held March 1-6,
 163 2020,¹ on the topic *Resiliency in Numerical Algorithm Design for Extreme Scale Simulations*.
 164 We should point out that, although error and resiliency was already quoted by J. von
 165 Neumann in his first draft report on EDVAC [260, P.1, Item 1.4], it became again a central
 166 concern for the HPC community in the late 2000' when the availability of the first exascale
 167 computers was envisioned for the forthcoming decades. In particular, several workshops
 168 were organized in the IESP (International Exascale Software Project) and EESI (European
 169 Exascale Software Initiative) framework [52].

170 The hardware/system resilience community has previously defined terminology related to
 171 how faults, errors, and failures occur on computing systems [18]. In this article our focus is
 172 less on the cause of an error (or the underlying fault), and more on how an error presents
 173 itself at the algorithmic level (or layer), impacting algorithms and applications. We thus
 174 simplify the terminology often used in the hardware resilience and fault-tolerance community
 175 by not using terms like soft error or hard error, and generally do not concern ourselves with
 176 the reproducibility of an error (e.g., transient, intermittent or permanent). This abstraction
 177 keeps the algorithmic techniques discussed herein general and applicable to a variety of fault
 178 models, current architectures, and hopefully of use in future technologies.

179 To this end, we broadly categorize errors presenting themselves to the algorithmic layer
 180 as either detected or undetected. Note that this categorization does not mean an error
 181 is undetectable but rather that when it reached the algorithmic layer it was not detected
 182 by earlier layers (e.g., hardware, operating system or middleware/system software). This
 183 suggests the algorithmic layer has the opportunity to detect a previously undetected error
 184 and, if possible, to deploy mitigation methods to make the algorithm resilient; effectively
 185 transforming an undetected error at the algorithmic layer into a detected error. This in turn
 186 may result in a failure if the algorithm is unable to handle it. For example, an undetected
 187 data corruption which results in an application accessing an incorrect memory address may
 188 be detectable by the algorithm but it may not be possible for the algorithm to recover what
 189 the original memory address was and it may be forced to fail. If the algorithm could not
 190 detect the corruption before accessing the memory region, this would conventionally end in a
 191 failure (e.g., *SIGSEGV* issued by the operating system).

192 Many computing-intensive scientific applications that are dependent on HPC performance

¹ <https://www.dagstuhl.de/en/program/calendar/semhp/?semnr=20101>



■ **Figure 1** A classification of error handling.

upgrades can end up with disrupted schedules because of lack of resilience. A typical example is related to current efforts towards exascale numerical weather prediction [31, 32]. On one side, regular upgrades in weather forecast models in operations at weather centres and their spatial resolution have gone hand in hand with expanding computational resources. On the other side, scientific and socioeconomic significance of forecasts crucially hinges on tight time-bound computing schedules and timely forecast dissemination, most notably for high-impact weather events. Current disk-checkpointing schedules still take up acceptable portions of forecast runtimes, but are hardly sustainable - indeed, they already saturate file systems bandwidth. In addition, many weather forecast codes feature preconditioned iterative solvers of linear systems with several hundred thousand unknowns, many thousand times per run. Such components represent vulnerable points in a context of increasingly frequent detected and undetected errors. Novel low-overhead solutions to enhance algorithmic fault-tolerance or provide higher-level system resilience are therefore in high demand in this and other fields where nonlinear dynamics is simulated.

In this article we take a different approach at the classification of errors in HPC systems. In general, we try to divide errors in two main groups, those that are detected and corrected by the hardware/system (which is the focus of Section 2) and those that are detected and sometimes corrected by the numerical algorithms (Section 3). However, the HPC resilience ecosystem is not black and white, but it rather shows a wide palette of greys in between, with multiple fault tolerance tools implemented at the middleware level that are assisted by the applications/algorithms and vice-versa. Figure 1 shows this wide range of different error classifications depending on how much effort is needed at the application/algorithmic level in order to detect/correct the error.

The first category we observe in the leftmost leaf of the tree (blue color) is the case

of errors that are both detected and transparently corrected by the hardware/middleware but without any intervention of the applications/algorithms. The clearest example would be a detectable and correctable error in the memory generated by a single bit flip. These types of errors are transparently corrected by the system without any knowledge at the application/algorithmic level that such error mitigation occurred. Other examples could be process replication, system-level checkpointing, process migration, among many others (see Section 2.1).

The second category is the case of errors that are detected at the hardware/system level and are mitigated at the system/middleware level (not at the algorithmic level) but with assistance from the application/algorithm (green color). The most clear example of this is application-based checkpointing libraries, which handle all or most of the data transfers between the compute nodes and the *Parallel File System* (PFS) independently from the application, but it gets hints from it to know what datasets need to be checkpointed and when should the checkpoint happen. Other relevant examples are fault tolerant message passing programming models and resilient asynchronous tasks. We divide these sections in those approaches that require just a minor addition in the application code versus those that require a complete change in the programming paradigm (see Section 2.2).

The other leaves of the tree (red color) correspond to those errors that cannot be corrected or mitigated at the hardware/system level and have to be mitigated by changing the algorithm or numerical methods to be able to tolerate those errors. We observe three different types of algorithms in this branch of the tree.

The first type of algorithms focuses on the mitigation of errors that have been detected (first red leaf from left to right), we call them error-aware algorithms. Please note that these algorithms are not in charge of detecting the errors but only of mitigating them. Also, it is important to notice that these algorithms do not depend on how the error was actually detected; it could be hardware/middleware detection as well as algorithmic detection, in the end the process of detection is irrelevant for the mitigation algorithm (see Section 3.2).

The second type of algorithms are those dedicated to the detection of errors that were not detected at the lower levels (fourth leaf from left to right). A good example would be *Silent Data Corruption* (SDC) errors that pass invisibly through the hardware but then can be caught at the algorithmic level using some numerical techniques (e.g., checksum). These algorithms do not try to mitigate the error per se but only detect it. Once the error has been detected, it can be passed to a error aware algorithm in order to attempt a correction/mitigation (see Section 3.1).

Finally, there also exist algorithms that can operate, tolerate and absorb errors without ever being aware that the error ever occurred (last leaf to the right); we called these, error oblivious algorithms. These are somehow similar to the very first (blue) category, in that the errors are transparently corrected/absorbed, see Section 3.3.

In the following sections we discuss algorithmic and application approaches to address these two categories of errors and distinguish how the approaches vary or are similar. Broadly speaking, the report is divided into two parts. In Section 2 and Section 3 we discuss the state-of-the-art in the areas of infrastructure and algorithms, while in Section 4 we propose possible areas of interest in future research.

2 System infrastructure techniques for resilience

In this section we describe the state-of-the-art of hardware and system level error detection and mitigation. As previously mentioned, we divide these methods in two categories, the ones

that mitigate the error in a completely transparent fashion, and those that require assistance from the algorithmic/application level. The following subsection, Section 2.1, concentrates on the methods falling in the first category. The second category is explored in Section 2.2.

2.1 Detected and transparently corrected errors

A wide range of errors can be detected and immediately corrected by various layers in the system, i.e., these errors become masked or absorbed and higher level layers do not have to be involved. The detection/correction mechanisms have an extra cost in terms of storage, processing and energy consumption.

Hardware reliability

At the hardware level several techniques exist to detect and correct errors. Most common examples are *Error Correcting Codes* (ECC) to detect and correct single bit-errors, *Cyclic Redundancy Checks* (CRC) error correction for network packets or RAID-1 (or higher) for I/O systems. A more comprehensive discussion of these features can be found in the report “Towards Resilient EU HPC Systems: A Blueprint” by Radojkovic et al. [207].

Operating system reliability

Operating Systems (OS) have certain capabilities to interact with architectural resilience features, such as ECC and machine check exceptions. OSs are mostly concerned with resource management and error notification. However, some advanced OS resilience solutions exist such as Mini-ckpts [92]. It is a framework that enables application survival despite the occurrence of a fatal operating system failure or crash. It ensures that the critical data describing a process is preserved in persistent memory prior to the failure. Following the failure, the OS is rejuvenated via a warm reboot and the application continues execution effectively making the failure and restart transparent. The mini-ckpts rejuvenation and recovery process is measured to take 3 s to 6 s and has a failure-free overhead of 3 % to 5 % for a number of key HPC workloads.

System-level checkpoint/restart

Berkeley Lab Checkpoint/Restart (BLCR) [125] is a system-level checkpoint/restart solution that transparently saves and restores process state. In conjunction with a *Message Passing Interface* (MPI) [95] implementation, it can transparently save and restore the process states of an entire MPI application. An extension of BLCR [257, 262, 263] includes enhancements in support of scalable group communication for MPI membership management, reuse of network connections, transparent coordinated checkpoint scheduling, a job pause feature, and full/incremental checkpointing. The transparent mechanism for job pause allows live nodes to remain active and roll back to the last checkpoint, while failed nodes are dynamically replaced by spares before resuming from the last checkpoint. A minimal overhead of 5.6% is reported in case migration takes place, while the regular checkpoint overhead remains unchanged.

The hybrid checkpointing technique [111] alternates between full and incremental checkpoints: At incremental checkpoints, only data changed since the last checkpoint is captured. This results in significantly reduced checkpoint sizes and overheads with only moderate increases in restart overhead. After accounting for cost and savings, the benefits due to incremental checkpoints are an order of magnitude larger than the overheads on restarts.

305 **Silent Data Corruption (SDC) protection**

306 FlipSphere [93] is a tunable, transparent *Silent Data Corruption* (SDC) detection and correc-
 307 tion library for HPC applications. It offers comprehensive SDC protection for application
 308 program memory using on-demand memory page integrity verification. Experimental bench-
 309 marks show that it can protect 50 % to 80 % of program memory with time overheads of 7 %
 310 to 55 %.

311 **Proactive fault tolerance using process or virtual machine migration**

312 Proactive fault tolerance [88, 187, 264] prevents compute node failures from impacting run-
 313 ning applications by migrating parts of an application, i.e., tasks, processes, or virtual
 314 machines, away from nodes that are about to fail. Pre-fault indicators, such as a significant
 315 increase in temperature, can be used to avoid an imminent failure through anticipation and
 316 reconfiguration. As computation is migrated away, application failures are avoided, which
 317 is significantly more efficient than checkpoint/restart if the prediction is accurate enough.
 318 The proactive fault tolerance framework consists of process and virtual machine migration,
 319 scalable system monitoring and online/offline system health analysis. The process-level live
 320 migration supports continued execution of applications during much of process migration
 321 and is integrated into an MPI execution environment. Experiments indicate that 1 s to 6.4 s
 322 of prior warning are required to successfully trigger live process migration, while similar
 323 operating system virtualization mechanisms require 13 s to 24 s. This error oblivious approach
 324 complements checkpoint/restart by nearly cutting the number of checkpoints by half when
 325 70% of the faults are handled proactively.

326 **Resiliency using task-based runtime systems**

327 Task-based runtime systems have appealing intrinsic features for resiliency due to the fault
 328 isolation they provide by design as they have a view of the task flow and dynamically schedule
 329 task on computing units (often to minimize the time to solution or energy consumption).
 330 Once an error is detected and identified by the hardware or the algorithm, the runtime system
 331 can limit its propagation through the application by reasoning about the data dependencies
 332 among tasks [176]. For example, one can envision the scenario where an uncorrectable
 333 hardware error is detected triggering the runtime system to dynamically redistribute the
 334 tasks to the remaining resources available.

335 Task-based runtime systems can also limit the size of the state needed to be saved to
 336 enable restarting computations, when an error is encountered [171, 172, 254]. In classical
 337 checkpoint-restart mechanisms, the size of the checkpoint can become very large for large-
 338 scale applications, and managing it can take up a significant portion of the overall execution.
 339 A task-based runtime system simplifies the identification of points during the application
 340 execution when the state size is small, since only task boundaries need to be considered for
 341 saving the state. Further, identification of idempotent tasks can greatly help task-based
 342 runtimes to further reduce the overheads by completely avoiding data backups specific to
 343 those tasks. Recent works on on-node task parallel programming models suggest that a
 344 simple extension of the existing task-based programming framework enables efficient localized
 345 recovery [200, 236, 238].

346 The checkpointing itself can also be achieved completely asynchronously [171, 172, 254].
 347 The runtime allows tasks to read data being saved, and only blocks those tasks that attempt
 348 to overwrite data being saved. Since the runtime system knows which data will soon be
 349 overwritten by some tasks, it can prioritize the writing of the different pieces so as to have

as little impact on the execution as possible. At the restarting point, the runtime also has all information to be able to achieve a completely local recovery. The replacement node can restart from the last valid checkpoint of the previously-failed node, while the surviving nodes can just replay the required data exchanges.

With the recent emergence of heterogeneous computing systems utilizing *Graphics Processing Units* (GPU), the task programming model is being used to offload computation from the *Central Processing Unit* (CPU) to the GPU. VOCL-FT [202] offers checkpoint/restart for computation offloaded to GPU using OpenCL [118]. It transparently intercepts the communication between the originating process and the local or remote GPU to automatically recover from ECC errors experienced on the GPU during computation. Another preliminary prototype design extends this concept in the context of OpenMP [42] using a novel concept for *Quality of Service* (QOS) and a corresponding *Application Programming Interface* (API) [89]. While the programmer is specifying the resilience requirements for certain offloaded tasks, the underlying programming model runtime decides on how to meet them using a QOS contract, such as by employing task-based checkpoint-restart or redundancy.

Resilience via complete redundancy

The use of redundant MPI processes for error detection has been widely analyzed in the last decade [55, 70, 208, 273]. Modular redundancy incurs high overhead, but offers excellent error detection accuracy and coverage with few to no false positive or false negatives.

Complete modular redundancy is typically too expensive for actual HPC workloads. However, it can make sense for certain subsystems such as parts of a PFS. The *Meta Data Service* (MDS) of a networked PFS is a critical single point of failure. An interruption of service typically results in the failure of currently running applications utilizing its file system. A loss of state requires repairing the entire file system, which could take days on large-scale systems, and may cause permanent loss of data. PFSs such as Lustre [67] often offer some type of active/standby fail-over mechanism for the MDS. A solution [128] for the MDS of the *Parallel Virtual File System* offers symmetric active/active replication using virtual synchrony with an internal replication implementation. In addition to providing high availability, this solution is taking advantage of the internal replication implementation by load balancing MDS read requests, improving performance over the non-replicated MDS.

Resilience via partial redundancy

Partial redundancy has been studied to decrease the overhead of complete redundancy [85, 234, 239, 240]. Adaptive partial redundancy has also been proposed wherein a subset of processes is dynamically selected for replication [108]. Partial replication (using additional hardware) of selected MPI processes has been combined with prediction-based detection to achieve SDC protection levels comparable with those of full duplication [37, 38, 188]. A Selective Particle Replication approach for meshfree particle-based codes protects the data of the entire application (as opposed to a subset) by selectively duplicating 1 % to 10 % of the computations within processes incurring a 1 % to 10 % overhead [57].

Resilience via complete and/or partial redundancy

RedMPI [91] enables a transparent redundant execution of MPI applications. It sits between the MPI library and the MPI application, utilizing the *MPI Profiling Interface* (PMPI) to intercept MPI calls from the application and to hide all redundancy-related mechanisms. A redundantly executed application runs with $r * m$ MPI processes, where m is the number of

MPI ranks visible to the application and r is the replication degree. RedMPI supports partial replication, e.g., a degree of 2.5 instead of just 2 or 3, for tunable resilience. It also supports a variety of message-based replication protocols with different consistency. Not counting in the need for additional resources for redundancy, results show that the most efficient consistency protocol can successfully protect HPC applications even from high SDC rates with runtime overheads from 0% to 30%, compared to unprotected applications without redundancy. Partial and full redundancy can also be combined with checkpoint/restart [85]. Non-linear trade-offs between different levels of redundancy can be observed when additionally using checkpoint/restart, since computation on non or less redundant resources is significantly less reliable than computation on fully or more redundant resources.

Interplay between resilience and dynamic load balancing

Scheduling of application jobs at the system level contributes to exploiting parallelism by placing and (dynamically) balancing the batch jobs on the local site resources. The jobs within a batch are already heterogeneous; yet, current batch schedulers rarely co-allocate, and most often only allocate, computing resources (while network and storage continue to be used as shared resources). Dynamic system-level parallelism can arise when certain nodes become unavailable (due to hard and permanent errors) or recover (following a repair operation). This can be exploited during execution by increasing opportunities for system-level co-scheduling in close proximity of jobs that exhibit different characteristics (e.g., co-scheduling a classical compute-intensive job in close proximity to a data-intensive job) and by dynamic resource reallocation to jobs that have lost resources due to failures or to waiting jobs in the queue.

2.2 Detected errors mitigated with assistance

In this section we focus on correction methods that need assistance from the upper layers in order to achieve resilience and correctness. It is important to note that there are multiple methods that offer assisted fault tolerance but some of them involve a few additional lines of code while others require rewriting the whole applications using a specific programming model. Therefore, we will divide this section into subsections depending on the programming and/or redesign effort that is required.

2.2.1 Correction with incremental redesign

As explained in Section 2.1, it is possible to perform system-level checkpointing without any feedback from the application or the algorithm or any upper layer. The issue with system-level checkpointing is that the size (and therefore the time and energy cost) of checkpointing is much larger than what is really required to perform a restart of the application. Thus, application-level checkpointing is an attempt to minimize the size of checkpoints to the minimum required for the application to be able to restart.

Performance modeling and optimization of checkpoint-restart methods

Research on simulation tools assessing the performance of certain checkpoint-restart strategies is presented in various publications [15, 73, 87, 165]. Different theoretical approaches are used and tools are developed that either simulate a fictional software or wrap an actual application.

A lot of work has been done to examine and model the performance of multilevel checkpointing approaches [28, 34, 156, 274]. Here, the parallel distribution of the snapshots as

well as the target storage system are considered as objectives for performance optimization. Asynchronous techniques are considered, such as non-blocking checkpointing, where a subset of processes are dedicated to manage the creation and reconstruction of snapshots [69, 219]. As a measure to saving storage and speeding up I/O, data compression is another subject that is considered in the literature as, e.g., by Di and Cappello [74], and in one of the case studies in Section 3.2.2.

Resilient checkpointing has been considered with the help of nonvolatile memory, as for instance implemented in PapyrusKV [154], a resilient key-value blob-storage. Other resilient checkpointing techniques include the self-checkpoint technique [245], which reduces common redundancies while writing checkpoints, or techniques reducing the amount of required memory through hierarchical checkpointing [182], or differential checkpointing [153].

Message logging

Message logging is a mechanism to log communication messages in order to allow partial restart as for example examined by Cantwell et al. [51]. While improving on basic checkpointing strategies, message logging-based approaches can themselves entail large overheads because of log sizes. The checkpointing protocol developed by Ropars et al. [213] does not require synchronization between replaying processes during recovery and limits the size of log messages. Other approaches combine task-level checkpointing and message logging with system-wide checkpointing [237]. This protocol features local message logging and only requires the restart of failing tasks. It is also possible to combine message logging with local rollback and *User Level Failure Mitigation* (ULFM) (Section 2.2.2) to improve log size [173].

Multilevel checkpointing libraries

Current HPC systems have deep storage hierarchies involving *High-Bandwidth Memory*, *Dynamic Random-Access Memory*, *Non-Volatile Memory*, *Solid-State Drives* and the PFS, among others. Multilevel Checkpointing libraries offer a way to leverage the different storage layers in the system through a simple interface. The objective is to abstract the storage hierarchy to the user, so that one does not need to manually take care of where the data is stored or the multiple data movements required between storage levels. Each level of checkpointing provides a different trade-off between performance and resilience, where usually lower levels use close storage that offers higher performance but limited resilience, and higher levels rely on stable storage (e.g., PFS), which is more resilient but slower. Mature examples of multilevel checkpoint libraries are SCR [183], FTI [28], CRAFT [226] and VeloC [189]. Both SCR and FTI provide support via simple interfaces for storing application checkpoint data on multiple levels of storage, including RAM disk, burst buffers, and the parallel file system. Both SCR and FTI provide redundancy mechanisms to protect checkpoint data when it is located on unreliable storage and can asynchronously transfer checkpoint data to the parallel file system in the background while the application continues its execution. In addition, FTI also supports transparent GPU checkpointing. Finally, VeloC is a merge of the interfaces of both FTI and SRC. Note that some of these libraries offer the option for keeping multiple checkpoints so that the application can roll-back to different points in the past if necessary.

Containment Domains

Containment Domains (CDs) provide a programming construct to facilitate the preservation-restoration model, including nesting control constructs, and durable storage [248]. The

following features are attractive for large-scale parallel applications. First, CDs respect the deep machine and application hierarchies expected in exascale systems. Second, CDs allow software to preserve and restore states selectively within the storage hierarchy to support local recovery. This enables preservation to exploit locality of storage, rather than requiring every process to recover from an error, and limits the scope of recovery to only the affected processors. Third, since CDs nest, they are composable. Errors can be completely encapsulated, or escalated to calling routines through a well-defined interface. We can easily implement hybrid algorithms that combine both preservation-restoration and data encoding.

Use cases include an implementation of a parallel resilient hierarchical matrix multiplication algorithm using a combination of ABFT (for error detection) and CDs (for error recovery) [16]. It was demonstrated that the overhead for error checking and data preservation using the CDs library is exceptionally small and encourages the use of frequent, fine-grained error checking when using algorithm based fault tolerance.

Application versioning

Global View Resilience (GVR) [64] accommodates APIs to enable multiple versioning of global arrays for the single program, multiple data programming model. The core idea is the fact that naive data redundancy approaches potentially store wrong applications states due to the large latency associated with error detection and notification. In addition to multiple versioning, GVR provides a signaling mechanism that triggers the correction of application states based on user-defined application error conditions. Use cases include an implementation of resilient Krylov subspace solvers [275].

Mitigating performance penalties due to resilience via dynamic load balancing

Detected and corrected errors induce variation in the execution progress of applications when compared to error-free executions. This can manifest itself as load imbalance. Many application-level load balancing solutions have been proposed over the years and can help to address this problem. We mention here a few available packages.

Available load balancing software includes Zoltan [252] that requires users to describe the workload across processes as a graph and offers an object oriented interface. Further we mention *Dynamic Loop Scheduling for Load Balancing* (DLS4LB) [54], a recently developed library for MPI applications that contains a portfolio of self-scheduling based algorithms for load balancing. StarPU [254] proposes support for asynchronous load-balancing [172] for task-based applications. The principle is to let the application submit only a part of its task graph, let some of it execute on the platform and observe the resulting computation balance. A new workload distribution can then be computed and the application is allowed to submit more of the task graph, whose execution can be observed as well. OmpSs [80] is an effort to extend OpenMP in order to support asynchronous execution of tasks including a transparent interface for hardware accelerators such as GPUs and FPGAs. OmpSs is built on top of the Mercurium compiler [251] and the nanos++ runtime system [249].

HCLib [271] is a task-based programming model that implements locality-aware runtime and work-stealing. It offers a C and C++ interface and can be coupled with inter-process communication models, such as MPI. Charm++ [151] features an automatic hierarchical dynamic load balancing method that overcomes the scalability limitation of centralized load balancing as well as the poor performance of completely distributed systems. Such a technique can be triggered dynamically after a failure hits the system and the workload needs to be redistributed across workers.

2.2.2 Correction with major redesign

The correction of some detected errors might have a strong impact of the algorithm that has to implement the mitigation. The mitigation design can be made more affordable if some components of the software stack have already some appealing features to handle such situations.

Resilience support in the Message Passing Interface (MPI)

Most MPI implementations by default are designed to terminate all processes when errors are detected. However, this termination occurs irrespective of the scope of the error, requiring global shut-down and restart even for local errors in a single process. This inherent scalability issue can be mitigated if MPI keeps all survived processes to continue and/or if restart overheads are reduced. The MPI community has proposed several recovery approaches, such as FA-MPI [127] or MPI-ULFM [41] to enable alternatives of global shut-down, as well as better error handling extensions, like MPI_Reinit [159], to reduce overhead and impact of failures. Among these approaches, MPI-ULFM is the most advanced and well known. It provides a flexible low-level API that allows application specific recovery via new error handling approaches and dynamic MPI communicator modification under process failures, although with significant complexities for the application developer using the new APIs. Several approaches have been proposed to mitigate this complexity by creating another set of library APIs built atop of MPI-ULFM [51, 99, 100, 225, 253]. However, as of now, in part due to its complexity when used on real-world applications and limited support in system software, MPI-ULFM as a whole has not been adopted in the MPI standard and hence is not readily usable for typical HPC application programmers. Nevertheless, various aspects of ULFM are in the process of standardization and will provide more mechanisms in MPI to build at least certain fault tolerant applications, starting with the upcoming MPI 4.0 standard.

Resilience abstractions for data-parallel loops

Data-parallel loops are widely encountered in N -body simulations, computational fluid dynamics, particle hydrodynamics, etc. Optimizing the execution and performance of such loops has been the focus of a large body of work involving dynamic scheduling and load balancing. Maintaining the performance of applications with data-parallel loops running in computing environments prone to errors and failures is a major challenge. Most self-scheduling approaches do not consider fault-tolerance or depend on error and failure detection and react by rescheduling failed loop iterations (also referred to as tasks). A study of resilience in self-scheduling of data-parallel loops has been performed using SimGrid-based simulations of highly unpredictable execution conditions involving various problem sizes, system sizes, and application and systemic characteristics (namely, permanent node failures), that result in load imbalance [241]. Upon detecting a failed node, re-execution is employed to reschedule the loop iterations assigned to the failed node.

A *robust Dynamic Load Balancing* (rDLB) approach has recently been proposed for the robust self-scheduling of independent tasks [180]. The rDLB approach proactively and selectively duplicates the execution of assigned chunks of loop iterations and does not depend on failure or perturbation detection. For exponentially distributed permanent node failures, a theoretical analysis shows that rDLB is linearly scalable and its cost decreases quadratically with increasing system size. The reason is that increasing the number of processors increases the opportunities for selectively and proactively duplicating loop iterations to

570 achieve resilience. rDLB is integrated into a dynamic loop scheduling library (DLS4LB, see
 571 Section 2.2.1) for MPI applications. rDLB enables the tolerance of up to $(P - 1)$ process
 572 failures, where P is the number of processes executing an application. For execution environ-
 573 ments with performance-related fluctuations, rDLB boosts the robustness of *Dynamic Loop*
 574 *Self-scheduling* (DLS) techniques by a factor up to 30 and decreases application execution
 575 time up to 7 times compared to their counterparts without rDLB.

576 Resilience extension for performance portable programming abstractions

577 With the increasing diversity of the node architecture of HPC systems, performance portability
 578 has become an important property to support a variety of computing platforms with the
 579 same source code while achieving a comparative performance to those programmed with the
 580 platform specific programming models. Today, Kokkos [82] and Raja [30, 250] accommodate
 581 modern C++ APIs to permit an abstraction of data allocation and parallel loop execution for
 582 a variety of runtime software and node architectures. This idea can be extended to express
 583 the redundancy of data and computation to achieve resilience while hiding the details of
 584 the data persistence and redundant computation. Recently, the resilient version of Kokkos
 585 was proposed for a natural API extension of Kokkos' data (memory space) and parallel
 586 loop (execution space) abstractions to (1) enable resilience with minimal code refactoring
 587 for the applications already written with Kokkos and (2) provide common interface to call
 588 any external resilience libraries such as VeloC [189]. The new software will be released in a
 589 special branch in <https://github.com/kokkos/kokkos>.

590 The resilience abstraction idea has also been applied to task parallel programming models
 591 such as Charm++ [151], HCLib [271], HPX [150], OmpSs [80] and StarPU [254] to integrate a
 592 variety of resilient task program execution options such as replay, replication, algorithm-based
 593 fault tolerance and task-based checkpointing. Task-based programming models indeed have
 594 a very rich view over the structure of the application computation, and notably its data,
 595 and have a lot of control over the computation execution, without any need for intervention
 596 from the application. Replaying a failed task consists of issuing it again with the same input,
 597 discarding the previous erroneous output, and replicating a task consists of issuing it several
 598 times with different output buffers and comparing the result. Dynamic runtime systems can
 599 then seamlessly introduce replay and replication heuristics, such as trying to run different
 600 implementations and/or computation units, without the application having to be involved
 601 beyond optionally providing different implementations to be tried for the same task.

602 The task graph view also allows for very optimized checkpointing [171, 172, 254]. In the
 603 task-based programming model, each checkpoint is a cut in the task graph, which can be
 604 expressed trivially within the task submission code, and only the data of the crossing edges
 605 need to be saved. Even better, the synchronization between the management of checkpoint
 606 data and application execution can be greatly relaxed. The transfer of the data to the
 607 checkpoint storage can indeed be started as soon as the data is produced within the task
 608 graph, and not only once all tasks before the checkpoint are complete. A checkpoint is
 609 then considered complete when all its pieces of data have been collected. It is possible that
 610 tasks occurring after the checkpoint may run to completion before the checkpoint itself is
 611 completed. All in all, this allows for a lot more time for the data transfers to complete, and
 612 lessens the I/O bandwidth pressure.

Software engineering approaches for resilience by design

Resilience design patterns [141,142] offer an approach for improving resilience in extreme-scale HPC systems. Frequently used in computer engineering, design patterns identify problems and provide generalized solutions through reusable templates. Reusable programming templates of these patterns can offer resilience portability across different HPC system architectures and permit design space exploration and adaptation to different (performance, resilience, and power consumption) design trade-offs. An early prototype [14] offers multi-resilience for detection, containment and mitigation of silent data corruption and MPI process failures.

3 Numerical algorithms for resilience

In this section, we focus on the handling of errors at the algorithmic level. We see three different classes of problems to tackle here: (i) detection of un-signaled errors (mostly bit flips and other instances of silent data corruption, Section 3.1), (ii) correction of errors that have been signaled but could not be corrected at the hardware or middleware layer (by error aware algorithms, Section 3.2), (iii) design of error oblivious algorithms that deliver the correct result even in the presence of (not too frequent) errors (Section 3.3).

In addition to correctness in the presence of errors, an important challenge in all our considerations is efficiency in terms of algorithm runtime. In this context, additional algorithmic components such as work stealing and asynchronous methods (where missing data are simply an extreme case of delay) have to be considered. We mention these methods when describing methods that can make use of such runtime optimizing measures.

3.1 Error detecting algorithms

In this section, we focus on mechanisms to numerically detect errors that have not been detected by the underlying system or middleware. We have identified several techniques that allow us to (likely) notice the occurrence of an error at several layers of numerical algorithms. Table 1 gives an overview of some detection techniques and the algorithmic components or numerical methods where they are applicable.

3.1.1 Exceptions

Exceptions are a way a program signals that something went wrong during execution. We consider the case where exceptions are caused by data corruption that can, for example, lead to division by zero or out-of-range access. Most programming languages support a way of handling exceptions. The algorithm programmer can register an exception handler that gets called whenever an exception occurs. If the error is recoverable, the exception handler will specify how best to continue afterwards. If the error is not recoverable, the program will be aborted. Exceptions are a straight-forward way to detect certain types of errors and can be applied to all numerical algorithms. However, they obviously only see a small subset of all possible errors and it is not trivial to decide when to use exceptions handlers in the light of a trade-off between correctness, robustness and runtime efficiency.

3.1.2 Checksums

Checksums could be used at the hardware or middleware layer to detect errors, but here we will discuss checksums as employed on the algorithmic layer where we have a more detailed

■ **Table 1** Numerical error detection: Overview of error detection techniques and numerical ingredients and methods where they are applied. Note that we mark a method as applicable only if it is or can be used in the respective algorithm itself, not only at lower level functionality, i.e., we do not mark checksums for multigrid as checksums are only used in the BLAS 2/3 kernels used as inner loops or in the GS/J/SOR smoothers.

	exceptions	checksum	constraints	tech error	multi resolution	redundancy
BLAS 2/3	×	×				×
Direct Solvers	×	×				×
Krylov	×		×	×		×
Multilevel / Multigrid	×			×	×	×
Domain Decomposition	×					×
GS/Jac/SOR	×			×		×
Nonlinear Systems	×			×		×
Time Stepping (ODEs)	×			×	(×)	×
PDEs	×	×	×	×	×	×
Quadrature	×		×	×	×	×

653 knowledge about the existence of numerical or algorithmic invariants. Checksum techniques
 654 have been used in various numerical algorithms. We list some examples below.

655 **BLAS 2/3:** Checksum encoding matrices, introduced by Huang and Abraham [137] requires
 656 (i) adding redundant data in some form (encoding), (ii) redesign of the algorithm to operate
 657 on the respective data structures (processing), and (iii) checking the encoded data for errors
 658 (detection). We ignore the recovery phase here and refer to Section 3.2. Checksums are used
 659 in FT-ScaLAPACK [267] for dense matrix operations such as MM, LU and QR factorization
 660 and more recently in hierarchical matrix multiplication [16]. Wu et al. give a good survey of
 661 checksum deployment in dense linear algebra [268].

662 **Gauss-Seidel/Jacobi/SOR and multigrid:** In [179], checksums are used to detect errors
 663 in the Jacobi smoother, the restriction and interpolation operators of a multigrid method
 664 solving a two-dimensional Poisson equation.

665 **Krylov subspace methods:** Tao et al. propose a new checksum scheme using multiple
 666 checksum vectors for sparse matrix-vector multiplication, which is shown to be generally
 667 effective for several preconditioned Krylov iterative algorithms [247]. Also [1, 227] use
 668 checksums for protection within the conjugate gradient (CG) algorithm.

669 **FFT:** Checksum can also be used in *Fast Fourier Transforms* (FFT)s similarly as in
 670 matrix-vector multiplication. Liang et al. [167] develop a new hierarchical checksum scheme
 671 by exploiting the special discrete Fourier transform matrix structure and employ special
 672 checksum vectors. Checksums are applicable to many important kernels such as matrix-
 673 matrix multiplication, but are costly. In addition, it can be difficult to specify a suitable
 674 threshold for ‘equality’ in the presence of round-off errors. For many numerical calculations
 675 such as scalar products, checksums are not applicable at all.

3.1.3 Constraints

In some applications, constraints for different types of variables are known. Examples are positivity constraints, conservation laws for physical quantities or known bounds for internal numerical variables.

Krylov subspace methods: Resilience was already of importance in the early days of digital computers. In the original PCG paper [132], Hestenes and Stiefel noticed that the reciprocal value of α (the step length) is bounded above (respectively, below) by the reciprocal of the smallest eigenvalues (respectively the inverse of the largest eigenvalue) of the matrix. The inequality involving the largest eigenvalue (for which in practice it may be cheaper to get an approximation) was used to equip PCG with error detection capabilities in [1].

Partial differential equations: Checking for bounds can be associated with minimal or extremely high cost depending on whether extra information has to be computed (such as eigenvalues of matrices) or not. Reliability is, in general, an issue as only those errors leading to violation of these constraints can be detected. An example of the use of problem-informed constraints can be found in [186]. In this work, the authors derive a priori bounds for the discrete solution of second-order elliptic PDEs in a domain decomposition setting. Specifically, they show that the bounds take into account the boundary conditions, are cheap to compute, general enough to apply to a wide variety of numerical methods such as finite elements or finite differences, and provide an effective way to handle faulty solutions synthetically generated.

3.1.4 Technical error information

In many numerical large scale applications, the main computational task involves the approximate computation of integrals, algebraic systems, systems of ODEs or PDEs. For all these problems, various types of error information such as residuals, differences between iterations, round-off error estimates and discretization error estimates can be used as indicators of errors either by their size or by monotonicity criteria. We give several examples from literature for different classes of numerical algorithms.

Krylov subspace methods: Round-off error bounds can be used in Krylov subspace methods. They fit in the general framework of round-off error analysis [133] and have been considered in the context of Krylov subspace methods in finite precision arithmetic [169, 178].

Vorst and Ye proposed a residual gap bound [256] (bound for the norm of the residual gap between the true and the computed residuals) based on round-off error analysis that was later used as a criterion for actual error detection in [1] when bit flips occur. The detection of errors in Krylov methods via violation of orthogonality is proposed in [63].

Multigrid: Calhoun et. al [50] apply a residual/energy norm-based error detection for algebraic multigrid. They use two criteria: (i) the reduction of the residual norm as a weak criterion and (ii) the reduction of the quadratic form

$$E(\mathbf{x}) = \langle A\mathbf{x}, \mathbf{x} \rangle - 2\langle \mathbf{x}, \mathbf{b} \rangle,$$

when solving the linear system $A\mathbf{x} = \mathbf{b}$ for symmetric positive matrices.

The quadratic for E calculated at level i during the down-pass of a V-cycle should be less than the energy calculated at level i during the down-pass of the next V-cycle.

When using the full approximation scheme residual norm reductions can also be verified at each level in the hierarchy of a multigrid-cycle. The structure of the full approximation scheme additionally provides smart recovery techniques utilizing its lower resolution approximations [11].

Time-stepping: For iterative time-stepping with spectral deferred corrections, monitoring the residual of the iteration can be used to detect errors in the solution vectors [119]. In the context of parallel-in-time integration with parareal, consecutive iterates are considered in [191] to detect errors in the solution vector. In [35], an auxiliary checking scheme in contrast to the original base scheme is used to detect and correct errors during implicit and explicit time-integration. Estimating the local truncation error with two different methods is used in [121] to implement a resilient, high-order Runge-Kutta method. This “Hot Rod” approach is then also used for error correction.

3.1.5 Multi-resolution

Multi-resolution means that information is available at different resolution levels, in terms of spatial discretization (PDE), time discretization (ODE and PDE), order of discretization (PDE in space and time), matrix dimensions (numerical linear algebra, multigrid), frequencies, and so on. This leads to a certain redundancy – not an artificially introduced, but an inherently available one. This redundancy can be used to detect discrepancies or anomalies and, hence, errors that could not be detected by the system. There are numerous examples for the mentioned problem classes, we outline one example in more detail here.

Sparse grids / Combination technique: Sparse grids [48] are one particular class of multi-resolution methods. There, via the use of hierarchical bases, certain structures often seen in d -dimensional data can be exploited to alleviate the curse of dimensionality, without a significant loss of accuracy. Sparse grids have been successfully used in a wide range of problem classes where spatial discretization plays a role, such as interpolation [145], quadrature [47, 109, 110], solvers for PDEs [124, 129], or machine learning tasks [104, 105, 201] (e.g., classification, regression, clustering, or density estimation). One particular incarnation of sparse grid methods is the so-called combination technique [117]. There, based on an extrapolation-style approach, a linear combination of a specific set of full, but very coarse-grid solutions is used to get a sparse fine-grid solution. The various coarse grid solutions can be obtained in a completely independent way, using (parallel) standard solvers. This opens the way to (1) a natural two-level parallelization and to (2) an easy and cheap detection of system undetected errors: Since we actually compute solutions for the same problem on different (i.e., differently discretized) grids anyway, we can use these to detect anomalies – just by comparing the available solutions. And the detection leads immediately to a mitigation strategy (see Section 3.2.2), since we can easily exchange coarse grids in case of errors, just by changing the combination pattern [8, 9, 123, 130, 134, 192]. Therefore, this is an example for a smart algorithm that is able to do both detection and mitigation.

Further examples are mentioned in Section 3.1.4 as multi-resolution typically comes with corresponding error estimates based on differences between solutions at different resolution levels: multigrid and parallel time stepping.

3.1.6 Redundancy

Redundancy is a strategy for error detection that can be applied to all of the numerical algorithms mentioned in Table 1. It covers two approaches. In the first approach computational resources may be replicated twice or thrice. Such instances are called DMR [144, 265] or TMR [223, 261]. In the second approach the computations are repeated twice or thrice on the same resource [17, 259]. An advantage of this approach is the flexibility at the application level. Note that the first approach costs more in space or resources, the second approach costs more in time.

The redundancy based error detection technique described in [33] relies on in-depth analysis of application and platform dependent parameters (such as the number of processors and checkpointing time) to formalise the process of both resource and computation replication. It provides a closed-form formula for optimal period size, resource usage and overall efficiency.

Ainsworth et. al [5] use replication of fault-prone components as an error detection technique in a multigrid method. Also error detection in the time stepping methods from [35] mentioned in Section 3.1.4 can be interpreted as redundancy based error detection.

The main disadvantage of replication is its cost in terms of performance, although recomputing only some instructions instead of the whole application lowers the time redundancy overhead [193]. However, redundancy in some calculations should in particular be considered as a possible strategy for error detection as in modern supercomputers the cost of arithmetic operations tends to decrease compared to communication time.

3.2 Error aware algorithms

In this section, we look at error correction techniques within an application. We assume that the application has been notified that part of the algorithm's data is corrupted or lost. In that context, mitigation or containment actions have to be undertaken at the algorithmic design level, where the appropriate actions depend on the data detection granularity and how the notification mechanism was activated. It is possible to design both lossy and lossless mitigation procedures that are tailored to the numerical algorithms under consideration.

In Section 3.2.1 we give a brief literature overview of ideas that can be used to complement numerical mitigation or containment procedures. Then, in Section 3.2.2 we offer a more detailed discussion of some recent successful attempts by presenting a few case studies in the context of the solution of *Partial Differential Equations* (PDE).

3.2.1 Error aware algorithms for the solution of linear systems

A wealth of literature already exists on various, mostly isolated ideas and approaches that have appeared over time. Checkpoint-restart methods are the most generic approaches towards resilience for a broad spectrum of applications, see Section 2.2.1 for an introduction. We first describe a general mental model to design resilient numerical algorithms independent of actual machine specifications that lead to what is nowadays referred to as *Local-Failure Local-Recovery* (LFLR) techniques. Then we move to 'classical' algorithm-based fault tolerance, which originally was developed to detect and correct single bit flips on systolic architectures devoted to basic matrix computations, see Section 3.1.2. Finally, we discuss a range of ideas and techniques not covered by the case studies below.

Local-failure local-recovery

As far back as a decade ago, an abstract framework was developed to separate algorithm design from unclear machine specifications, see also Section 2.2.1. The idea of a selective reliability model as introduced by Hoemmen [45,135] is machine-oblivious and highly suitable for algorithm design for machines with different levels of (memory) reliability. It has led to the concept of *Local-Failure Local-Recovery* (LFLR) [253]. This model provides application developers with the ability to recover locally and continue application execution when a process is lost. In [253], Teranishi and Heroux have implemented this framework on top of MPI-ULFM (Section 2.2.2) and analyzed its performance when a failure occurs during the solution of a linear system of equations.

809 Original algorithm-based fault tolerance with checksums

810 The term *Algorithm-Based Fault Tolerance* (ABFT) was originally coined in conjunction
 811 with protecting matrix operations with checksums to handle bit flips [136], mostly assuming
 812 exact arithmetic calculation for detection and mitigation. (See Section 3.1.2 for a more
 813 detailed discussion on checksums). The main drawback of checksums is that only limited
 814 error patterns can be corrected and its robust practical implementation in finite precision
 815 arithmetic can be complicated to tune to account for round-off errors. A second drawback
 816 is that the checksum encoding, detection and recovery methods are specific to a particular
 817 calculation. A new scheme needs to be designed and proved mathematically for each new
 818 operation. A further drawback is to tolerate more errors, more encoded data is needed, which
 819 may be costly both in memory and in computing time.

820 ABFT concepts have been extended to process failures for a wide range of matrix
 821 operations both for detection and mitigation purposes [44, 62, 79, 147, 269] and general
 822 communication patterns [149]. ABFT has also recently been proposed for parallel stencil-
 823 based operations to accurately detect and correct silent data corruptions [58]. In these
 824 scenarios the general strategy is a combination of checkpointing and replication of checksums.
 825 In-memory checkpointing [147] can be used to improve the performance. The main advantage
 826 of these methods is their low overhead and high scalability.

827 In practice, the significance of a bit flip strongly depends on its location, i.e., which bit in
 828 the floating point representation is affected. Classical ABFT has been extended to take into
 829 account floating point effects in the fault detection (checksums in finite precision) as well
 830 as in the fault correction and to recover from undetected errors (bit flips) in all positions
 831 without additional overhead [181].

832 Iterative linear solvers

833 Iterative linear solvers based on fixed point iteration schemes are, in general, examples of
 834 error oblivious algorithms, as described in Section 3.3. The convergence history of the scaled
 835 residual norm observed within the iterative scheme often resembles the curves displayed in
 836 Figure 2. In this case the iterative scheme is a multigrid method, as in [115, 138]. The peaks
 837 in the residual occur after data has been lost and when the iterations are allowed to restart
 838 with some form of replacement of the lost data. In the simplest case, the lost data may just
 839 be re-initialized with the value of zero, and recovery techniques to obtain better solutions
 840 are discussed in Section 3.2.2.

841 It can be seen that, depending on when in the course of the iteration a small portion of
 842 the approximate solution suffers from an error, we observe a delay in convergence, directly
 843 proportional to an increase in runtime. In the case where errors appear too often, the solver
 844 might not recover and other mitigation actions might have to be considered.

845 Explicit recovery at the algorithmic level from undetected errors have been studied
 846 for iterative linear solvers [195]. In contrast to restarting, a number of algorithm based
 847 recovery strategies have been proposed, including approximate or heuristic interpolation
 848 methods [2]. An approach of exactly recovering the state of the iterative solver before the
 849 node failure has been investigated for the *Preconditioned Conjugate Gradient* (PCG) and
 850 related methods [164, 196]. This also includes studying scenarios with multiple simultaneous
 851 node failures [197] and scenarios where no replacement nodes are available [194].

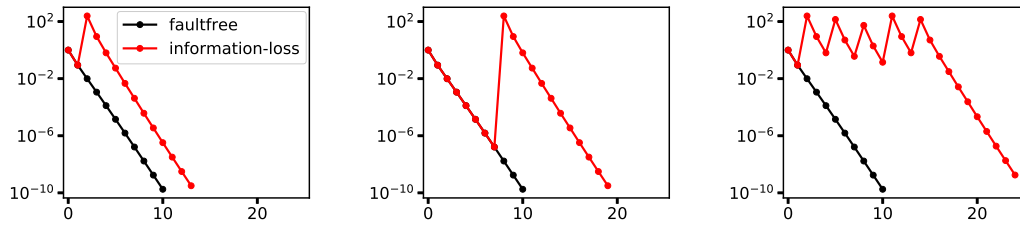


Figure 2 Convergence history of the residual norm as a function of the iteration count for three examples of information loss. From left to right: early, late, and multiple times

Approximated recovery and restart in sparse numerical linear algebra

For matrix computations, eigensolvers or basic kernels such as iterative linear system solvers, some recovery ideas rely on forming a small dimensional linear algebra problem where the inputs are the still valid data and the unknowns are the lost/corrupted ones. The outcome of this procedure is subsequently used to replace the lost/corrupted data and the numerical algorithm is somehow started again from that meaningful initial guess. The recovery procedure is tailored to the actual numerical algorithm. As an example, consider a fixed point iteration scheme for a linear system and suppose the lost data are entries of the iterate vector, the most dynamically evolving data in this computational scheme. Matrix entries of the iteration scheme related to the lost data, as well as some neighbouring entries, serve to build the left-hand side of a linear problem (either a linear system or a least-square problem) while the right-hand side is built from valid data. The solution of this small problem is then used to replace the corresponding lost entries of the iterate vector. The complete, updated vector is taken as a new initial guess when restarting the fixed point iteration. If the data is not corrupted too often the classical convergence theory still applies and because the new initial guess incorporates updates from the calculations performed before the error was detected, the global convergence rate is not strongly affected. The method described in adaptive recovery techniques for extreme scale multigrid in Section 3.2.2 is an example application of this technique.

For numerical schemes based on nested subspace search, such as Krylov subspace methods, closely related techniques have been successfully applied both for eigensolvers and linear solvers that further exploit the sparsity structure of the matrices to reduce the computational cost associated with the recovery procedure. At the cost of a light checkpoint performed once when starting the linear solver (mostly the matrix and the right-hand side vector in case of linear system solution) this mitigation approach has no overhead if the data is not corrupted during the solution computation. We refer to [2, 3, 161] for some illustrations on those numerical remedies in a parallel distributed memory framework and to [146] where these ideas are exploited for a lower granularity of data loss in a task-based runtime system. See Section 2 for references relevant to task-based runtime systems.

We also note that these ideas can be extended to hybrid iterative/direct numerical schemes, that have a domain decomposition flavor, where the recovery procedure can be enriched with additional features of the parallel numerical scheme such as redundancy or properties of the preconditioners [4]. They can also be extended to the time domain in the context of multilevel parallel-in-time integration techniques [229].

886 3.2.2 Error aware algorithms for the solution of partial differential 887 equations

888 The ideas introduced above in Section 3.2.1 are application agnostic but naturally apply to
889 linear systems arising from the discretization of a PDE. In that latter case, more information
890 from the underlying PDE can be closely tailored to intrinsic features of solvers such as
891 multigrid. In this section we discuss some research works on mitigation and containment
892 that exploit the properties of PDEs to aid the recovery techniques. We also present some
893 mitigation processes that are only relevant in the PDE setting.

894 Adaptive recovery techniques for extreme scale multigrid

895 Some of the most efficient solvers of PDE, such as parallel geometric multigrid methods
896 [114, 143], can be based on the exchange of ghost layers in a non-overlapping domain
897 partitioning. This automatically leads to a redundancy in interface data between subdomains
898 that in turn permits the design of an efficient two-step recovery strategy for iterative solvers.
899 This is of particular interest in large-scale parallel computations. When each subdomain
900 is large, then the ratio between the data on its surface and the volume data in its interior
901 becomes small.

902 When a processor fails, the information within one or several subdomains is lost. For
903 the recovery and continued solution, the redundant ghost layer information is used in a
904 first step, to recover locally either Dirichlet- or Neumann-type data for the subdomains.
905 The global problem can then be formulated in two partitions, the outer healthy subdomain
906 and the inner faulty subdomain, where the recovery must reconstruct the lost data. Both
907 subproblems must be bi-directionally coupled via the interface and the corresponding ghost
908 layers of unknowns.

909 After re-initialization, the corrupted and reinitialized data could pollute the solution
910 globally, meaning that the locally increased error in the faulty domain can spread globally and
911 thus also affect the healthy subdomain. In order to avoid this pollution, the communication
912 between the healthy and faulty sub-problems is interrupted during the second step of the
913 recovery process. In the second step, we continue with the original iterative solver restricted
914 to the healthy sub-problem and select a suitable one for the faulty one. After some number
915 of asynchronous iteration steps both sub-problems are reconnected, see [138], and the global
916 iterative solver strategy is resumed. Note that the reconnecting step is mandatory for the
917 convergence of the iterative solver. The tearing step separating the subdomains is mandatory
918 to preserve the accuracy of the dynamic data in the healthy sub-problem, and without this
919 step the corrupted data from the faulty sub-domain pollutes the global solution. Of critical
920 importance for the performance of the method are the accuracy of the faulty sub-problem
921 solver at re-connection time and the time spent in the recovery mode. In the faulty domain,
922 the lost data can be initialized with 0, or, alternatively, compressed checkpointed data can be
923 used as described in the following section on compression techniques for checkpoint-restart.
924 Note, however, that with straight-forward compression techniques, compressed checkpoint
925 data will only be useful to recover the low frequency components in the faulty domain. If the
926 local recovery is performed with multigrid, then the low frequencies are in any case cheap to
927 recover, as compared to the cost of recomputing the lost high frequency components.

928 The accuracy within a multigrid strategy can be easily controlled by a hierarchical sum
929 of weighted residuals [216]. The overhead cost for the a-posteriori error indicator is quite
930 small compared to the overall solver cost. Having an estimate for the algebraic error in both
931 sub-problems at hand, the re-connection step is determined automatically. To speed up the

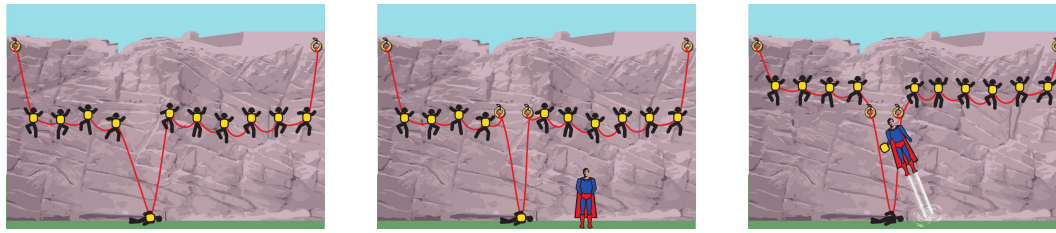


Figure 3 Illustration of the steps in the adaptive recovery technique for extreme scale multigrid. Left: A detectable error occurred. Middle: The communication between the healthy and faulty sub-domains is interrupted. Right: The original iterative solver restricted to the healthy domain continues while another suitable solver is asynchronously used in the faulty domain. Once the solution in the faulty domain reaches a certain accuracy, the communication between the domains is re-enabled.

time which is spent in the recovery, a so-called ‘superman strategy’ is applied [138], see also Figure 3 for an illustration. More resources compared to the situation before the fault are allocated to the faulty sub-problem. A short recovery phase in combination with carefully selected re-coupling criteria then guarantees a highly efficient fault-tolerant solver.

Of special interest is a massively parallel multigrid method as base solver. In combination with the tearing and intersection approach for the recovery, it results in a hybrid approach. In case of a Stokes-type system, yielding after discretization a saddle point problem, the strategy can either be applied on the positive definite Schur complement for the pressure or, as it was done in [139], on the indefinite velocity-pressure system. In that case an all-at-once multigrid method with an Uzawa-type smoother acting on both solution components turns out to be most efficient, see [78]. Numerical and algorithmic studies including multiple faults and large-scale problems with more than $5 \cdot 10^{11}$ degrees of freedom and more than 245000 cores have been demonstrated [138, 139]. The automatic re-coupling strategy is found to be robust with respect to the fault location and size and also handling multiple fault. In many scenarios a complete recovery can be achieved with almost no increase in runtime and while maintaining excellent parallel efficiency.

Adaptive mesh refinement, load balancing, and application level checkpointing

Adaptive Mesh Refinement (AMR) functionality and load balancing require similar data linearization- and transfer functionality as is needed for application level checkpointing. This is exploited in the waLBerla framework [24, 221, 222] that features an object oriented design for composing coupled multiphysics simulation software. waLBerla’s load balancing is based on routines to transfer simulation data between processors so that functionality to serialize, pack, send, and unpack all relevant data is already available as a by-product of the AMR functionality. Note that the waLBerla software architecture imposes this structure for Eulerian mesh based data as well as for Lagrangian particle-based models and it canonically extends to coupled Eulerian-Lagrangian multiphysics models. For this to work transparently, the routines for migrating simulation data must be suitably encapsulated. Then this functionality can be used to write user level checkpoints either on disk or in memory. Note that writing checkpoints will inevitably imply overheads in memory consumption and communication time, but that restoring a checkpoint is cheap, since it initially only requires re-activating the redundantly stored data. This is especially true when in-memory checkpointing is used as explored and analyzed in [156]. The simple restoration of checkpointed data may of course lead to load imbalance, but the functionality to redistribute load is also available as part

of the parallel AMR functionality. In this sense, user-level checkpointing can be based in a natural, efficient, and straightforward way on the functionality of parallel AMR algorithms combined with load balancing functionality.

Compression techniques to accelerate checkpoint-restart for Krylov-MG solvers

Compressed checkpointing is a possibility to improve the efficiency of classical checkpoint-restart schemes, both in terms of the overhead to generate the checkpoints and to recover the data if an error occurs. The added efficiency mainly comes from a reduced memory footprint which is beneficial for communication and storage. It is particularly efficient if the compression method is tailored to the target application. As an example, in-memory compressed checkpoints combined with LFLR (see Section 3.2.1) for iterative linear solvers, e.g., multigrid preconditioners in Krylov schemes, are described below.

Lossy Compression: As already mentioned in Section 3.2.2, paragraph ‘Approximated recovery and restart’, initially only the dynamical data, i.e., the approximate solution, are protected. Lossy compression allows a balance between the accuracy of the discretization error of the assembled system and the numerical error within the solver. Specifically in [10], the SZ library [72, 166, 246] is employed, which 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 rate and higher compression time, which is crucial in terms of resilience overhead.

Note that multigrid can be interpreted as a lossy compression technique in itself, with a number of mathematical peculiarities that need consideration [115]. 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 a lower-resolution representation of the iterate can then be stored as a compressed checkpoint. Conversely, the multigrid prolongation (coarse-to-fine grid interpolation) operator is used to decompress the data. With only small additional computations, the multigrid hierarchy can also be used for error detection.

Recovery: Several recovery techniques can be devised [10]. As a baseline approach checkpoint-restart is mimicked and the global iterate is simply replaced with its decompressed representation, independently of the compression strategy. The second proposed approach follows the LFLR strategy and re-initializes only the local data that is lost on faulty computing nodes by using checkpoint data stored on neighbouring computing nodes. Contrary to the first approach, this is mostly 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 processes except those sending the checkpointed data. As a worst-case fallback when the backup data is not sufficient, a third recovery approach is established, which is still mostly local. Here, an auxiliary problem is solved iteratively with boundary data from the neighbouring computing nodes. This is similar to the adaptive recovery techniques for extreme scale multigrid from above or the approximated recovery and restart of Section 3.2.1. An auxiliary problem is constructed, either by domain decomposition overlap or the operator structure, and solved with an initial guess based on the checkpoint data to accelerate the iterative recovery phase. Experiments show that this approach can almost always restore the convergence speed of the fault-free scenario independently of the used backup technique, only the number of additional local recovery iterations varies. For more details, we refer to [10].

1010 Resilience with sparse grids

1011 Resilience can be added on various abstraction levels of the algorithm. For PDE problems
1012 one traditionally adds resilience on the level of linear algebra operations, on the solver level
1013 for linear/non-linear equations, or on the time-stepping algorithm. However, this may in
1014 some cases not be coarse-grained enough to minimize the overhead of resilience techniques,
1015 especially when errors occur rarely. In [123, 129, 130, 134, 192, 199] the authors demonstrate
1016 a fault-tolerant framework for solving high-dimensional PDEs that applies fault tolerance
1017 on top of the individual PDE solver. The framework boosts the scalability of black-box
1018 PDE solvers while making it simultaneously resilient to faults by applying the sparse grid
1019 combination technique. In this technique the PDE simulation is distributed over many coarse
1020 grids, which can be processed in parallel. At regular intervals the results of these grids are
1021 combined to obtain the final sparse grid result. In presence of faults the affected grids can be
1022 neglected and an alternative combination scheme is calculated via an optimization routine.
1023 If too many grids are lost, the last combination result serves as an in-memory checkpoint
1024 to recompute the required grids. In [192] it is shown that this lossy recovery provides very
1025 good results even with high error frequencies. At the same time the parallel efficiency is only
1026 slightly affected.

1027 Adaptive mesh refinement

1028 Adaptive refinement techniques in combination with finite element methods are well estab-
1029 lished for fault-free computations. In terms of fault tolerance, this means that in addition to
1030 the assembled linear system, the geometric mesh structure must be protected. This requires
1031 the reconstruction of the data structures containing the mesh hierarchy. For the use of
1032 multigrid or multilevel methods, we also need to recover multiple levels of adaptive grid
1033 refinement after a fault has occurred. The recovery process must take into account the
1034 intra-grid as well as the inter-grid data dependencies.

1035 We refer to [233] for a parallel adaptive multigrid method that uses a sophisticated
1036 dynamic data structures to store a nested sequence of meshes and the iterative evolving
1037 solution. Stals demonstrates that it is possible to implement a fault recovery procedure
1038 that builds on the original parallel adaptive multigrid refinement algorithm [232] in the case
1039 of a fail-stop fault. It is assumed that a copy of the coarsest grid can always be accessed
1040 after a fault has occurred, i.e., it is stored off the processor. The challenge in recovering
1041 an adaptively refined grid is that the mesh distribution changes during any load balancing
1042 procedures, i.e., the local information that was available during the original refinement process
1043 will have been modified or removed. Nevertheless it is demonstrated that the neighbouring
1044 healthy processors contain enough intact information so that the necessary structure can be
1045 recovered to pass into the refinement routine. In the case of uniform refinement, the original
1046 multilevel grid is recovered. In the case of an adaptively refined grid, enough of the structure
1047 is recovered to re-establish the correct communication pattern allowing the solution process
1048 to run to completion, but potentially with reduced accuracy. The neighbouring healthy
1049 processors will only contain enough information to guide the refinement around the edge of
1050 the recovered subgrid. Further refinement within the interior of the recovered subgrid may
1051 be required to improve the accuracy of the solution.

1052 These techniques were implemented with minimal disruption to the original code. An
1053 example of one the few necessary modifications is that in the original code, communication
1054 was used to ensure that the elements were refined in the appropriate order to avoid degenerate
1055 grids. In the resilient version of the the code that communication had to be removed as the

1056 refinement was restricted to the faulty processor.

1057 3.3 Error oblivious algorithms

1058 In this section, we give examples of algorithms that are error oblivious in the sense that they
 1059 can recover without assistance from errors that do not occur too frequently. For example,
 1060 many fixed point iterative solvers are able to execute to completion if, e.g., a bit flip error
 1061 occurs in the solution vector. However, every error likely increases the execution time of
 1062 the algorithm. We thus define two quality criteria for error oblivious algorithms and use to
 1063 assess the examples in the remainder of this section: (i) correctness, and (ii) efficiency in
 1064 terms of execution time.

1065 Finding an algorithm that fulfills (i) and can also compete against error aware algorithms
 1066 as described in Section 3.2 remains an open problem.

1067 Error oblivious usually means that an error slowly ‘leaves the system’ during several
 1068 iterative sweeps over the data. Error mitigation in error aware algorithms, on the other hand,
 1069 requires specific measures to correct the error, and can only be applied when the error has
 1070 been detected on a hardware, middleware or algorithmic layer, but removes the disturbance
 1071 of the calculation process by the error immediately.

1072 We do not expect the error oblivious algorithms to be impervious to all types of errors.
 1073 An iterative method may be not error oblivious if the error changed the matrix entries. This
 1074 concept is defined as selective reliability, see Section 3.2.1.

1075 3.3.1 Gossip based methods

1076 A potentially interesting alternative in large-scale parallel environments that does not require
 1077 any explicit error detection mechanisms utilizes gossip-based methods and their inherent
 1078 resilience properties. Such algorithms by nature build up redundancy in the system and can
 1079 thus can efficiently recover automatically from various types of faults/errors without any
 1080 need to explicitly detect them. In particular, Gansterer et al. have studied and extended the
 1081 resilience of gossip-based aggregation and reduction methods [56, 101, 190]. Based on these
 1082 building blocks, they have developed and analyzed several more complex resilient numerical
 1083 algorithms, such as orthogonalization methods [101, 102], eigensolvers [235], and least squares
 1084 solvers [205].

1085 While the strong resilience properties and execution-time robustness of these approaches
 1086 are promising, there is a certain price in terms of basic runtime compared to classical
 1087 deterministic numerical high performance algorithms. It remains to be investigated whether
 1088 they can be competitive in a fault-prone, but otherwise classical system with global view
 1089 and centralized control. Their competitiveness can be expected to increase significantly if
 1090 some of these classical properties have to be weakened at the extreme scale.

1091 3.3.2 Fixed-point methods

1092 We view fixed-point methods as methods that converge globally when certain conditions are
 1093 satisfied. For example, the Jacobi iterative schemes will converge for any initial guess if the
 1094 matrix is diagonally dominant. Fixed-point based iterative methods are by design resilient
 1095 to bit flips. However, the convergence delay can be significant. Anzt et al. [12, 13] propose
 1096 techniques improving the cost-robustness with little overhead.

1097 A class of numerical algorithms that by design have properties attractive for resilience
 1098 are asynchronous iterative methods [23, 29, 39, 40, 97, 230, 231, 242]. In order to avoid

misunderstandings, we point out that this class of methods is unrelated to the idea of asynchronous dynamic load balancing [155] as addressed in Section 2.1. Instead, asynchronous iterative methods, stemming from the concept of chaotic iterations [60], are fixed-point methods that seek the solution of a problem by independently updating subdomains – which can be subdomains in the geometric sense, subsets, or individual components of the solution approximation – according to some fixed-point linear or nonlinear iterative scheme. A particularity of the asynchronous methods is that the independent updates neither adhere to a specific update order, nor synchronize in terms of a handshake with other updates, but still converge globally in the asymptotic sense. In particular, these methods are robust with respect to some subdomains being updated at a much lower pace as each update just uses the most recent non-local information available. In that sense, asynchronous solvers can have good performance in unreliable environments where messages can be dropped or processes can become unresponsive for limited time. Also, in cases where messages are corrupted (and corruption can be detected), an asynchronous solver can simply drop such a message. In cases where processes remain unresponsive, a mechanism is still needed to recover that process and its state, but the remaining processes can continue computing unchanged. Therefore, asynchronous methods are somehow error oblivious.

With the increasing cost of global synchronizations, and the attractive properties concerning fine-grained parallelization and resilience against communication delays and errors, asynchronous methods have gained attention in particular for numerical computations [243]. Chow et al. [65, 66] developed an asynchronous algorithm for generating incomplete factorizations, Coleman et al. [68] further improved this algorithm by employing measures that reduce the runtime overhead when encountering errors. More general is the idea of asynchronously updating subdomains in Schwarz decompositions. In particular asynchronous restricted additive Schwarz methods and asynchronous optimized Schwarz methods have been identified to combine algorithm-inherent resilience with scalability on pre-exascale hardware architectures [83, 103, 112, 175, 270].

Independently, asynchronous multilevel methods have been proposed and analyzed under the name Fully Adaptive Multigrid method [214]. Here the multigrid smoothing process is executed asynchronously so that it can be employed for concurrent operations on different levels of the mesh hierarchy. The iteration is executed in a Southwell style [228] and is controlled by efficient hierarchical error estimators [216]. The parallel implementation [215] will automatically correct errors. More recently, asynchronous methods have been proposed for nonlinear multi-splitting [244] and eigenvalue computations like Google's Pagerank algorithm [157]. More recently, also the idea of asynchronously solving coarse-grid error correction equations has been investigated, leading to an asynchronous multigrid algorithm [266]. While case studies reveal attractive properties, these newly developed asynchronous iterative methods (such as asynchronous multigrid) are not fixed-point iterations, and developing a convergence theory for those algorithms remains a challenge.

3.3.3 Krylov subspace solvers

A comprehensive overview about the use of selective reliability with Krylov methods in the presence of bit flips is given in James Elliott's PhD thesis [86]. Elliott evaluates the CG and GMRES solvers with the algebraic multigrid preconditioner, see also [10] for a more recent study. Coleman et al. [68] consider Krylov subspace solvers in combination with the incomplete ILU algorithm ParILUT. In [84] Elliot et al. investigate the effect of bit flips on the convergence of GMRES and propose strategies for minimizing the numerical impact.

The authors of [31] present a monotonicity-based fault detection and correction procedure

for a Generalized Conjugate Gradient Krylov solve and perform tests with manual fault injection. While the solver manages to converge even with large amounts of corrupted data, the basic recovery procedure speeds up convergence with minimal detection and correction overhead.

In [218] the authors use a slightly different terminology and call their method numerically self-stabilizing, a term which originates in the context of distributed systems [77]. They introduce two error oblivious [77] iterative linear solvers: one for the steepest descent and one for conjugate gradient. In the latter case, they consider necessary conditions for conjugate gradient to converge. Those conditions are borrowed from non-linear conjugate gradient [276] and are maintained in a correction step (typically performed every other ten iterations). The correction step does not explicitly correct errors, but re-computes quantities such as the residual at regular intervals. Therefore, we classify these methods as error oblivious instead of error aware.

3.3.4 Domain decomposition

In [116] Griebel and Oswald use probabilistic analysis to model the effect of errors on the convergence of the classical overlapping Schwarz algorithm. They conclude that this method does indeed converge in the presence of errors. Glusa et al. [113] mention that asynchronous domain decomposition methods are by definition fault-tolerant. In [184,210,211], the authors discuss resiliency of a task-based domain decomposition preconditioner for elliptic PDEs. By leveraging the domain decomposition approach, the problem is reformulated as a sampling problem, followed by a regression-based solution update. The regression is formulated and implemented such that it is oblivious to corrupted samples. The authors combine this algorithmic approach with a server-client implementation based on ULFM, see Section 2.2.2. They show promising results of this approach in terms of resiliency to missing tasks, corrupted data and hardware failure.

3.3.5 Time stepping

In [119], iterative time-stepping using spectral deferred corrections are shown to be error oblivious at the cost of more iterations for the affected time-step. With error-estimators in place, time-integration techniques like Runge-Kutta methods will repeat the calculation of a time-step with smaller step sizes, if errors in the solution vectors are relevant [61]. This type of algorithms is resilient against errors in the solution vector of the new time step. Repeating the new time step with a reduced time step size is not the optimal measure in case of an error where repeating the step with the same time step size would be more efficient, but it leads to correct results.

4 Future directions

In the final section we focus on the future direction of resilient algorithms. We highlight what changes need to be made to current infrastructures to support the goals proposed by algorithm and application developers. Furthermore, we list those algorithms that are likely to come to the forefront as resiliency plays a more important role in the cost-benefit analysis of extreme scale simulations. And we mention some numerical methods that are yet to be fully explored in the context of resilient algorithms.

4.1 Systems in support of resilient algorithms

We propose that resiliency will only be obtained by a multilayered approach incorporating operating systems, file systems, communication, programming models, algorithms, applications and education. In terms of the layers covered by infrastructure, the goal is to increase systems and delivered performance while keeping the detectable errors in the upper algorithm based layers constant. We refer the reader to the recently published report by Radojkovic et al. [207] for an overview of the needs of the next generation HPC systems.

4.1.1 Error correcting codes

Poulos et al. [204] propose hardware ECC assistance that can pass error syndrome information through to an application and use this to fix detected errors. When an ECC hardware error occurs that results in a *Detectable, but Uncorrectable Error* (DUE), the ECC hardware generates a syndrome which is a byproduct of the error detection. For many ECC schemes, a syndrome that corresponds to a DUE can be used to generate a list of possible corrections, one of which is taken to be the original uncorrupted data. In this work, the authors show that this set is relatively small, meaning that the set of potential values for an application to search for their correct answer (before corruption) is also small. They also study the error value distribution and show that for certain classes of problems it can be easy to identify obviously wrong answers. For the application studied in [204], work was done to correct a hydrodynamics application using conservation laws and average of neighbor cells. This work requires changes to the hardware error reporting techniques and modification to the operating system to determine which application observed the DUE and pass it to an interrupt handler.

4.1.2 Improving checkpoint/restart

Independent of any additions, changes or new developments in the algorithmic or the system area, checkpoint/restart will remain a necessary component for any system. For one, no other technique can provide the needed resilience against full system outages; further, checkpoint/restart is also needed for developers to deal with limited job execution times and possible migration between systems or debugging purposes at large scale.

Improving classical checkpoint/restart for homogeneous systems

Observing the necessity of checkpoint/restart makes it critical to further optimize, enhance and support efficient checkpoint/restart mechanisms—even on classical, homogeneous systems—and provide users with library based solutions for core checkpoint/restart functionality. In particular, the following avenues should be pursued to optimize checkpoint/restart.

- Use additional algorithmic techniques to be able to reduce checkpoint frequency.
- Reduce data to be written to disk by eliminating redundancy and possibly compressing checkpoint information. Note that suitable data compression will typically require user-level knowledge, suitable interfaces must be provided.
- Overlap/Offload checkpoint operations to allow for asynchronous checkpoint/restart operations.
- Integrate checkpoint/restart with novel programming approaches to minimize checkpointable state.
- Keep the restart requirements local to the neighbour nodes of the failed node.

- 1228 ■ Localize checkpoint data to own or localized nodes. This could be supported by local
1229 non-volatile memory, as targets for checkpoint data. While this has the potential to
1230 reduce communication, as it avoids remote data transfers, it may require additional
1231 hardware support to retrieve data from non-functional nodes, e.g., by accessing data
1232 through fast JTAG-like interfaces.
- 1233 ■ In memory checkpointing.
- 1234 ■ Exploit user-level knowledge for serializing, packing, compressing data, see e.g. how exist-
1235 ing AMR functionality [156] can be exploited for efficient checkpointing in Section. 2.2.1.

1236 **Checkpoint/Restart for heterogeneous systems**

1237 In addition to classical checkpoint/restart for homogeneous systems, node-local check-
1238 point/restart support for heterogeneous systems will help containing error and failure
1239 propagation. Such support may be provided transparently to the application by the under-
1240 lying infrastructure, such as GPU drivers or task-based environments, or exposed in the
1241 programming model, such as OpenMP Offload [76].

1242 **4.1.3 Scheduler and resource management**

1243 Support for resilience, especially at the workflow-level, has a direct impact on resource
1244 management in HPC systems and hence requires new developments in this area as well.

1245 **Node-level parallelism**

1246 With increasing node-level parallelism, the impact of OS noise (typically caused by un-
1247 predictable interrupts) becomes even more important. Therefore, dedicated node-level
1248 resources are needed to exclusively run the OS and minimize the impact of OS noise on the
1249 multi-threaded application running on the other cores.

1250 **Adaptive system and application load balancing**

1251 The batch scheduler needs to adaptively balance the system load onto the available resources,
1252 via seamless application migration. While the application needs to adapt to the capabilities
1253 of the newly allocated resources, if different from the original allocation, without incurring
1254 performance penalties. The former has typically been implemented via checkpointing and
1255 process migration [174]. The latter has typically been implemented for applications that can
1256 adjust their granularity, e.g. from finer to coarser, depending on resource availability either
1257 triggered by the application or the system [46]. When exposing and expressing parallelism
1258 in applications, in addition to accounting for and matching the multiple levels of hardware
1259 parallelism (nodes, sockets, cores), the decomposition granularity needs to be flexible to
1260 support evolvability and malleability and allow for adaptive load balancing at the application
1261 and system levels.

1262 **Adaptive resource management**

1263 The batch scheduler in conjunction with the distributed runtime system employed by the
1264 application (e.g., MPI, Charm++, HPX) needs to support resources errors/failures and
1265 recover them without terminating the applications in the process. This approach should work
1266 both with rigid and moldable applications as well as with evolving and malleable applications.

■ **Table 2** Properties of numerical algorithms fostering or helping resilience

categories	solvers	discretization
redundancy	×	×
replication	×	
hierarchical methods	×	×
mixed precision	×	×
error control	×	×
locality-emphasizing schemes		×
asynchronous methods	×	×
embarassingly parallel	×	
stochastic > deterministic		×
iterative vs direct solvers	×	
matrix-free / low memory footprint	×	×

4.2 Programming models with inherent resiliency support

Certain applications and algorithms may naturally be resilient against errors. This may lend them as natural candidates for asynchronous parallel execution (via asynchronous many-task programming). While this mitigates the challenges associated with bulk synchronous parallel execution, asynchronous parallel execution may influence, in the presence of silent errors, the convergence rate of the numerical algorithms and might lead to incorrect results.

Programming model and runtime support for resilience can offer transparent handling of errors and failures or can assist the application in handling them. Consistent programming model support for resilience based on realistic error/failure models is needed to properly handle such events with low overhead. Higher-level abstractions for programming resilient applications are needed to help with error/failure handling complexities and to offer reuse of concepts and codes across applications.

4.3 Future directions for the solution of partial differential equations

In this section, we focus on discretizations for linear and non-linear partial differential equations as well as solvers for the resulting discrete and sparse systems of equations. We introduce a list of algorithmic properties that we found are, or can be, contributing to the resilience of the algorithms described in Section 3. Table 2 lists these properties and indicates where we found relevant examples of how they can foster resilience for either linear or non-linear solvers or for spatial or time discretization. In the following subsections we describe these examples in more detail and highlight the several (mutually related) properties that could be of interest in the context of resilient algorithms.

4.3.1 Redundancy and replication

A failure that is not fixed by the system (hardware and middleware) typically results in a loss or corruption of data. To tackle this problem, redundancy techniques can be used to detect and recover from data corruption and data loss. The performance of these algorithms is usually measured in the amount of memory and computational overhead they entail, the detection rate of errors, the rate of false-positives they achieve, and the accuracy of the recovery. Optimizing these performance indicators should be of main concern for future algorithm design. One existing class of algorithms that apply redundancy are

multiresolutional techniques such as multigrid and the sparse grid combination technique described in Section 3.2. They inherently add redundancy through the hierarchical structure. Sparse grid combination techniques calculate the same solution on different anisotropic grids. The coefficients of the combinations of the components grids can be recalculated if one or more nodes are lost due to faults. This redundancy of the component grids allows the algorithm to obtain an alternative approximation of the solution. However, if a component grid is distributed on too many nodes, then the approximation will fail if a fault occurs on any one of those nodes. Another class of algorithms add redundancy through recomputation with different models and configurations such as in ensemble or multifidelity techniques. A more straight-forward approach is to directly add redundancy through replication of certain algorithmic paths, cf. the following subsection on recalculation techniques.

Depending on the underlying architecture, replication can be a competitive option to increase detected and undetected error robustness. If computation speed significantly outpaces memory access and communication, each operation can be executed multiple times while the data is still accessible in the RAM. This can be used for redundancy-based sanity checks of low-level operations or even for checksum-like approaches.

Overlapping data in parallel algorithms can serve as a starting point for mitigation, albeit not for detection. In the case studies explored in Section 3.2.2, these are applied to elliptic PDEs, though an extension to other models should be feasible. Furthermore by even increasing the ghost layer size and thereby adding extra redundancy, other reconstruction possibilities might become possible. This could already be taken into account during the domain partitioning process.

4.3.2 Hierarchy and mixed precision

Hierarchical discretizations have proven to be advantageous in various respects. Related notions are multi-resolution or multi-level discretizations, but also (recursive) sub-structuring in the engineering nomenclature of the *Finite Element Method* (FEM). Built into the hierarchy are problem-inherent information and structures that are well-suited for modern hierarchy-based solvers. In FEM, for example, hierarchical bases carry information about both location and frequency, which leads to a special built-in redundancy that can be exploited for error detection (see Section 3.1). Therefore, from a resilience perspective, hierarchy should be a core paradigm for discretization design. This applies irrespectively of whether the hierarchical bases are formulated in the spatial (h) or the order (p) sense.

From a solver perspective, multigrid methods for elliptic and parabolic PDE problems are relevant approaches towards resilient numerical algorithms. They inherently act on different granularities, representations, scales, and levels and can be used to quantify differences between these levels. For local recovery, local multigrid methods are highly efficient, especially when they can be accelerated with the superman strategy [138]. Additionally the low-resolution duplicates can be used for some kind of approximate recovery or minimal rollback like re-application of the smoother on a specific level in a multigrid scheme. Detection of errors within multigrid is often possible due to algebraic relations or on the basis of hierarchical multi-grid-inherent error estimates [11, 139, 216], which hold true inside such schemes. As stated in Section 4.3.1, the inherent redundancy incorporated in these algorithms is also beneficial.

Mixed-precision arithmetics are typically used within the numerical solver parts to speed-up computations. However, the discretization can enable the flexibility to store data at varying precision. Examples for this are hierarchical approaches such as hierarchical bases, where a function value is stored as a hierarchical surplus only. As another example, the

usage of wavelets in multiresolutional analysis can serve. In both cases, contributions of higher levels typically require less accuracy, as only the most significant bits contribute to the overall point values.

4.3.3 Error control

For many numerical methods, a wide range of classical a priori and a posteriori error estimation techniques are available, see among many others [6, 22, 122, 152, 160, 206, 216], which constitute the basis of many adaptive numerical algorithms.

Adaptive time discretization methods are the state of the art for ODE solvers, while, for PDE solvers, spatial adaptivity techniques are also widely used. Local time step adaptation is feasible in the framework of so called local time stepping or multirate approaches, where different components of the system can have different time step sizes, see [43, 53, 94, 106, 217, 220, 224], which are however still far from mainstream for most applications. For PDE solvers, local spatial adaptivity techniques are also very common [20, 21], but their incorporation in operational applications is still a research topic, see e.g. [36, 163, 185, 203, 255] for developments concerning oceanography and numerical weather forecasting.

The error estimations on which all these methods rely on also constitute the basis of an error detection mechanism, since some undetected errors, like bit flips on significant floating point digits, will result in errors exceeding the allowed error tolerances. To some extent, these techniques are also examples of ABFT or error oblivious approaches, since bit flips and other silent errors occurring during the computation of the solution at the next time step or on a refined mesh could be automatically corrected by the repeated computations triggered by the error threshold violation. Furthermore, silent errors in the data at the current time or mesh level could be identified by the failure of the time step or mesh refinement to correct the error.

Combined with other ABFT strategies, adaptive discretization strategies based on error estimators can be a powerful and so far rather underrated tool for protecting a simulation from undetected errors in the solution vectors. On the other hand, error estimators should not be used as a black box for resiliency purposes. Indeed, errors can lead to severe over-resolution or, potentially, even under-resolution in space or time and the error estimators themselves could be affected by undetected errors.

As seen in Section 3.1.4, some iterative solvers for the solution of linear systems have invariants, such as monotonicity for Krylov solvers. These properties can be put to good use in devising resilience strategies, for example activating an additional restart of the Arnoldi procedure as soon as an increase in the residual norm is observed.

The idea of interval arithmetic is to compute bounds of intervals that always contain the exact result [7, 158]. Probabilistic methods for rounding error estimation [71, 96, 98, 198, 258] require several executions of arithmetic operations with different perturbations or different rounding modes (for instance three executions for Discrete Stochastic Arithmetic [81]). With both approaches, the comparison of several computed results enables one to control rounding errors (or detect and mitigate actually wrong results).

4.3.4 Locality, asynchronicity and embarrassingly parallelism

One important aspect of resilient algorithms is error confinement as global dependencies propagate errors to other processors and complicate recovery. Locality-emphasizing numerical algorithms achieve this by limiting dependencies to local areas or completely removing them. Consequently, error mitigation can be limited to a local subdomain. Typical examples for

these schemes are domain decomposition, which splits the domain into several subareas, and classical discretization schemes such as finite elements, finite differences and finite volumes.

As mentioned in Section 3.2.1, domain decomposition schemes such as additive Schwarz methods, or substructuring-inspired FETI [90] or also the fully adaptive multigrid method [214] are naturally asynchronous and resilient to message loss. In this context, we use the term asynchronous primarily in the sense of reducing the time synchronicity in parallel computations – from communication-avoiding schemes via a reduction of synchronization points up to vastly decoupled schemes. Using this inherent property, a failure in a subdomain would result in a message loss that does not hinder convergence in other subdomains, because a global wait for a message update and synchronization are not necessary. In addition, asynchronous methods may better adapt to heterogeneous processors and networks than their synchronous counterparts as it has been shown in the context of Grid computing [19,59]. Both the localized and asynchronous approaches, achieve their impact through a decoupling of computations. Going further in this direction leads to embarrassingly or nearly embarrassingly parallel algorithms. These represent a group of algorithms where it is relatively easy to decouple subproblems in time or space. The subproblems can therefore be calculated completely independently, and errors do not propagate to other subproblems. Examples of such methods are Monte Carlo simulations and computations with the sparse grid combination technique. Since it is expected that only a few tasks will encounter errors and the scheduling is automatically balancing the load, the overall execution time does not suffer too much. Future algorithmic design should therefore aim at increasing asynchronicity and locality to move towards embarrassingly parallel problems.

4.3.5 Stochastic

Stochastic methods can be superior to deterministic methods when it comes to resilience. Stochastic methods do not require the program to take a deterministic path, faulty parts can be neglected or exchanged easily by other results. A popular example are Monte Carlo methods where we sample randomly in the computation domain and can simply neglect failed samples. Ensemble methods are examples where different instances or models of a concrete problem setting are computed. Even if one of these computation fails, the ensemble computation can still return a – maybe slightly less accurate – result. Stochastic elements can therefore help the future algorithm design to reduce the dependencies on specific results of the computation. These methods, however, need to be evaluated not just by highlighting their resilience properties, but also taking into account the cost of a single run: if a single run is expensive to complete, simply discarding it might be impractical.

4.3.6 Iterative methods

Iterative solvers may be viewed as inherently more robust than direct solvers because they do not compute their solution using a pre-defined sequence of numerical operations as direct solvers typically do. Indeed, by their nature, they perform a sequence of operations to update and improve their current approximation. If an error is encountered during computation, the probability of deleting this error or at least its effect may be higher than in a direct solver. Especially fixed-point-based methods (domain decomposition, relaxation, ...) may be viewed as inherently resilient as they have the property to always converge to the correct solution independent of the initial state (global convergence). Some errors may induce a low influence on convergence speed and can thus be safely ignored. In other cases, a restart – optionally with recovery techniques – may be employed to ensure both resilience and efficiency in terms

1433 of runtime.

1434 4.3.7 Low memory footprint – matrix-free

1435 The classical approach to represent linear operators as sparse matrices produces large
1436 amounts of static data which has to be restored upon failure. Checkpoint-restart approaches
1437 feature high memory cost, naturally multiples of the storage needed for the solution vector.
1438 Algorithmic alternatives to checkpoint-restart require possibly complicated or costly re-
1439 assembly. Matrix-free methods do not represent the operators as static data in the first place.
1440 Therefore, large sparse matrix data structures do not have to be restored upon failure as they
1441 are computed on the fly anyway. Extreme-scale applications will benefit from matrix-free
1442 approaches due to their low memory footprint, also in terms of runtime, (due to high memory
1443 access cost) and higher limits for the overall problem size [25,27].

1444 In addition to saving memory and, therewith, reducing the risk of memory corruption,
1445 matrix-free methods can also be combined with automatic code generation [162] in a stencil-
1446 based approach, i.e., for finite difference methods on uniform structured grids. In such cases,
1447 the matrix entries may be ‘hard wired’ into code, such as 5-point stencils for Laplace’s equation.
1448 Automatic code generation provides a means to increase resiliency in the code generator or
1449 domain specific language and, thus, facilitate resilience aware software development.

1450 For finite element methods, one can use local assembly kernels [26]. Here, the trade-off
1451 between computation and storage and, in the future, resilience is relevant in particular for
1452 higher order elements.

1453 4.4 The final mile: towards a resilient ecosystem

1454 The future directions described above will provide critical enhancements towards providing
1455 resilient computation for numerical simulations. Alone, however, they are insufficient, as
1456 they must be embedded in the larger ecosystem and in the efforts to make that ecosystem
1457 support such novel resilience approaches. This requires another set of crucial developments.

1458 4.4.1 Tools to support resilience software development

1459 Developers will need the right tools to support their algorithmic efforts. These tools, as
1460 they exist today, are often designed without faults and errors in mind and, therefore, do
1461 not sufficiently support the development of resilient systems. In particular, we identified
1462 three areas in which enhanced tool support for resiliency is needed: a) introspection to help
1463 track errors and failures along with their root causes, b) validation through controlled fault
1464 scenarios to enable targeted testing of new error mitigation features, and c) transformation
1465 to transparently add error and failure checks into codes.

1466 Tools for introspection

1467 Introspection is critical to ensuring early error detection and the timely activation of correction
1468 and mitigation mechanisms throughout the various layers of the software ecosystem.

1469 *System Monitoring:* Knowing about the health state of a system requires monitoring it
1470 and understanding its behavior. Future work needs to focus on scalable system monitoring,
1471 real-time analyzes of system monitoring data, and autonomous decision making on corrective
1472 actions for self-aware resilient systems. In order to gain a deeper understanding, types of
1473 monitored data should be homogenized across system and sites, and, if possible, sanitized
1474 logs should be available to the community.

1475 *Application and Data Structures Monitoring:* Applications need to automatically monitor
 1476 their performance and correctness with the use of tools. The tools can be developed in
 1477 abstraction, at the compiler-level, or at the runtime-level.

1478 **Tools for validation**

1479 Currently, there are no standard tools to test the correctness and performance of resilient
 1480 algorithms under undetected errors and fault. This is due to a lack of fault injection tools
 1481 that reflect realistic situations. DeBardeleben et al. [120] have developed a hardware error
 1482 simulator tool to understand the behavior of numerical algorithms under faulty hardware with
 1483 a great accuracy, but this approach cannot evaluate the execution time of resilient algorithms
 1484 at scale. Vendors provide fault injection tools [126, 148] for better execution efficiency,
 1485 compromising the accuracy of the hardware behavior. Compiler approaches or other in-house
 1486 error injections [49, 107] could allow the program to execute as efficiently as the original
 1487 binary, but the correctness is further compromised. There are also tools that can analyze
 1488 an application's vulnerability very quickly but do not actually produce the application's
 1489 faulty output. One technique for this, DiSCvar [177], uses algorithmic differentiation and
 1490 exposes how changes to each variable impact output results. It is important to note that
 1491 these techniques do not actually produce that corrupted output. Hence, they are very fast
 1492 but they may not be useful to developers looking to explore precisely how corruption changes
 1493 their application. It is likely that a combination of these techniques, which identify most
 1494 critical regions of an application coupled with fault injection at those locations, may serve as
 1495 a good compromise between the two techniques.

1496 Any novel approaches that fill the gap between the accuracy and execution efficiency of
 1497 error injections will facilitate the code development of resilient algorithms, and the new tools
 1498 should be built with the existing continuous integration infrastructure. Such tools likely
 1499 require hardware knowledge that is considered intellectual property by the semiconductor
 1500 vendors. However, efforts which explore this space using open hardware technologies (RISC-V,
 1501 Sparc, etc.) can shed light on this space but may be of varying usefulness when application
 1502 developers look to understand how their applications will perform on hardware that has not
 1503 been fault injected at the register transfer or microcode level.

1504 **Tools for code transformation**

1505 Compilers are able to generate binaries with resilience capability as suggested in the work
 1506 by [209]; the generated binary instruments redundant computation, register allocations
 1507 to enable error detection and correction during program execution. The recent work by
 1508 Lin [170] leverages LLVM to generate SIMD instructions to perform redundant computation
 1509 and verification. Source-to-source code transformation has been proposed to enable triple
 1510 modular redundancy in loops [168] and automatic instrumentation of checkpointing [212].
 1511 Similarly, this idea can be extended to redundant threading for error mitigation, facilitated
 1512 with OpenMP-like programming language extension [140]. These approaches automatically
 1513 introduce resilience with some performance penalty, preventing the users from selective
 1514 adaptation of resilience for performance optimization, and these redundant computations are
 1515 benefited from the memory hierarchy, preventing doubling (or tripling) of the execution time.

1516 In addition to such specific systems that support the addition of resilience to existing
 1517 codes, automated generation of code, e.g., via *Domain Specific Languages* (DSL) can help
 1518 with the transparent support of resilient computation. Examples for this can be stencil
 1519 generators, as already discussed in Section 4.3.7.

4.4.2 User/Programmer education

According to the system log study by [75], many application job failures are triggered by the mistakes of the users such as script errors and program bugs including excessive file and thread creations. This means that better software engineering practices and training of users should be pursued with similar efforts to the deployment of resilience strategies.

The Exascale Computing Project (ECP) by the US DOE has made a substantial investment on educating tools, software engineering and HPC system usage for a variety of the users. Additionally, the scientific and mathematical library teams in the ECP have introduced software engineering policies [272] to improve the software quality, documentation and testing process for better interoperability and composability of multiple library packages. This activity, though not directly relevant to resilience, will gradually help to reduce application errors and failures for large scale HPC systems.

5 Conclusions

This article presents a snapshot of current research on resilience for extreme scale computing. It has grown out of the Dagstuhl seminar 20101 held March 1-6, 2020, bringing experts from the field together on the topic *Resiliency in Numerical Algorithm Design for Extreme Scale Simulations*. This seminar became a starting point to develop a synthesis between the system perspective on resilience and the algorithmic perspective.

While resilience is undoubtedly an issue for extreme scale computing, it is less clear what algorithms on the user or application level can contribute to mitigate faults. The seminar provided ample room to discuss these topics and thus became the starting point for this article. Many diverse aspects were found to be relevant, that require a holistic and multidisciplinary approach involving different and complementary scientific communities.

In particular, it clearly appeared that a fundamental distinction lies in whether faults are detected or not, and if they are not automatically detected, whether they are detectable. If they are, algorithms can often be developed to detect errors and in a second stage to correct them. It was found that some algorithms are naturally tolerant against faults or have the intrinsic feature to be error oblivious. They can thus be naturally applied on a system subject to errors.

Besides redundancy and checkpointing as classical techniques to mitigate faults, new algorithm-based resilience techniques have been developed for several classes of numerical algorithms. This includes linear algebra and solvers for partial differential equations, two classes of algorithms that are prominent in many scientific workloads on supercomputers. Some of these mitigation methods show remarkable success in the sense that faults can be compensated algorithmically by recovery procedures with only little extra cost in time or in silicon. On the other hand it also becomes clear that integrating such techniques in a computational infrastructure is still facing many obstacles. This includes the still poorly defined interface between user-level fault mitigation techniques and system level functionality, as, it is, e.g., necessary to reliably and quickly detect a device (core, memory, ...) failure on a large parallel machine.

Despite its breadth, the article is far from being comprehensive. The selection of topics is a subjective overview of current research in the field of resilience for extreme scale computing and it delivers an outlook into possible and promising future research topics and solutions.

1563 — References —

- 1564 **1** Emmanuel Agullo, Siegfried Cools, Emrullah Fatih-Yetkin, Luc Giraud, Nick Schenkel, and
1565 Wim Vanroose. On soft errors in the Conjugate Gradient method: sensitivity and robust
1566 numerical detection - revised. Research Report RR-9330, Inria, March 2020. URL: <https://hal.inria.fr/hal-02495301>.
1567
- 1568 **2** Emmanuel Agullo, Luc Giraud, Abdou Guermouche, Jean Roman, and Mawussi Zounon. Nu-
1569 merical recovery strategies for parallel resilient Krylov linear solvers. *Numerical Linear Algebra*
1570 *with Applications*, 23(5):888–905, August 2016. URL: <https://hal.inria.fr/hal-01323192>,
1571 doi:10.1002/nla.2059.
- 1572 **3** Emmanuel Agullo, Luc Giraud, Pablo Salas, and Mawussi Zounon. Interpolation-restart
1573 strategies for resilient eigensolvers. *SIAM Journal on Scientific Computing*, 38(5):C560–C583,
1574 2016. URL: <https://hal.inria.fr/hal-01347793>, doi:10.1137/15M1042115.
- 1575 **4** Emmanuel Agullo, Luc Giraud, and Mawussi Zounon. On the resilience of parallel sparse hybrid
1576 solvers. In *HiPC 2015 - IEEE International Conference on High Performance Computing*,
1577 Bangalore, India, December 2015. URL: <https://hal.inria.fr/hal-01256316>.
- 1578 **5** Mark Ainsworth and Christian Glusa. Is the multigrid method fault tolerant? the multilevel
1579 case. *SIAM Journal on Scientific Computing*, 39(6):C393–C416, 2017.
- 1580 **6** Mark Ainsworth and J Tinsley Oden. *A posteriori error estimation in finite element analysis*,
1581 volume 37. John Wiley & Sons, 2011.
- 1582 **7** G. Alefeld and J. Herzberger. *Introduction to interval analysis*. Academic Press, 1983.
- 1583 **8** Md Mohsin Ali, Peter E. Strazdins, Brendan Harding, and Markus Hegland. Complex scientific
1584 applications made fault-tolerant with the sparse grid combination technique. *International*
1585 *Journal of High Performance Computing Applications*, 30(3):335–359, 2016.
- 1586 **9** Md Mohsin Ali, Peter E. Strazdins, Brendan Harding, Markus Hegland, and Jay W Larson.
1587 A fault-tolerant gyrokinetic plasma application using the sparse grid combination technique.
1588 In *Proceedings of the 2015 International Conference on High Performance Computing &*
1589 *Simulation (HPCS 2015)*, pages 499–507, Amsterdam, The Netherlands, July 2015.
- 1590 **10** Mirco Altenbernd, Nils-Arne Dreyer, Christian Engwer, and Dominik Göldeke. Towards
1591 local-failure local-recovery in PDE frameworks. In *Proceedings of HPCSE’19*. Springer, 2020.
1592 accepted.
- 1593 **11** Mirco Altenbernd and Dominik Göldeke. Soft fault detection and correction for multigrid. *The*
1594 *International Journal of High Performance Computing Applications*, 32(6):897–912, November
1595 2018. doi:10.1177/1094342016684006.
- 1596 **12** Hartwig Anzt, Jack Dongarra, and Enrique Quintana-Ortí. Fine-grained bit-flip protec-
1597 tion for relaxation methods. *Journal of Computational Science*, 36:100583, 2019. URL:
1598 <http://www.sciencedirect.com/science/article/pii/S1877750316303891>, doi:<https://doi.org/10.1016/j.jocs.2016.11.013>.
1599
- 1600 **13** Hartwig Anzt, Jack Dongarra, and Enrique S. Quintana-Ortí. Tuning stationary iterative
1601 solvers for fault resilience. In *Proceedings of the 6th Workshop on Latest Advances in Scalable*
1602 *Algorithms for Large-Scale Systems*, New York, NY, USA, 2015. Association for Computing
1603 Machinery. doi:10.1145/2832080.2832081.
- 1604 **14** Rizwan Ashraf, Saurabh Hukerikar, and Christian Engelmann. Pattern-based modeling of
1605 multiresilience solutions for high-performance computing. In *Proceedings of the 9th ACM/SPEC*
1606 *International Conference on Performance Engineering (ICPE) 2018*, pages 80–87, Berlin,
1607 Germany, April 9-13, 2018. ACM Press, New York, NY, USA. doi:10.1145/3184407.3184421.
- 1608 **15** Rizwan A. Ashraf and Christian Engelmann. Performance efficient multiresilience using
1609 checkpoint recovery in iterative algorithms. In *European Conference on Parallel Processing*,
1610 pages 813–825. Springer, 2018.
- 1611 **16** B. Austin, E. Roman, and X. Li. Resilient matrix multiplication of hierarchical semi-separable
1612 matrices. In *Proceedings of the 5th Workshop on Fault Tolerance for HPC at eXtreme Scale*
1613 *(FTXS)*, pages 19–26, Portland, Oregon, June 2015. doi:10.1145/2751504.2751507.

- 1614 17 Todd M. Austin. DIVA: A reliable substrate for deep submicron microarchitecture design. In
 1615 *Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture,*
 1616 *MICRO 32, IEEE Computer Society, Washington, DC, USA, 1999.*
- 1617 18 A. Avizienis, J. . Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of
 1618 dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*,
 1619 1(1):11–33, Jan 2004. doi:10.1109/TDSC.2004.2.
- 1620 19 J. M. Bahi, S. Contassot-Vivier, and R. Couturier. Evaluation of the asynchronous iterative
 1621 algorithms in the context of distant heterogeneous clusters. *Parallel Computing*, 31(5):439–461,
 1622 2005.
- 1623 20 Wolfgang Bangerth, Carsten Burstedde, Timo Heister, and Martin Kronbichler. Algorithms and
 1624 data structures for massively parallel generic adaptive finite element codes. *ACM Transactions*
 1625 *on Mathematical Software (TOMS)*, 38(2):1–28, 2012.
- 1626 21 Wolfgang Bangerth and Rolf Rannacher. *Adaptive finite element methods for differential*
 1627 *equations*. Birkhäuser, 2013.
- 1628 22 Randolph E. Bank and R Kent Smith. A posteriori error estimates based on hierarchical bases.
 1629 *SIAM Journal on Numerical Analysis*, 30(4):921–935, 1993.
- 1630 23 G. Baudet. Asynchronous iterative methods for multiprocessors. *Journal of the Association*
 1631 *for Computing Machinery*, 25:226 – 244, 1978.
- 1632 24 Martin Bauer, Sebastian Eibl, Christian Godenschwager, Nils Kohl, Michael Kuron, Christoph
 1633 Rettinger, Florian Schornbaum, Christoph Schwarzmeier, Dominik Thönnies, Harald Köstler,
 1634 et al. walberla: A block-structured high-performance framework for multiphysics simulations.
 1635 *Computers & Mathematics with Applications*, 2020.
- 1636 25 Simon Bauer, Daniel Drzisga, Marcus Mohr, U Rüe, Christian Waluga, and Barbara Wohlmuth.
 1637 A stencil scaling approach for accelerating matrix-free finite element implementations. *SIAM*
 1638 *Journal on Scientific Computing*, 40(6):C748–C778, 2018.
- 1639 26 Simon Bauer, Markus Huber, S Ghelichkhan, Marcus Mohr, Ulrich Rüde, and B Wohlmuth.
 1640 Large-scale simulation of mantle convection based on a new matrix-free approach. *Journal of*
 1641 *Computational Science*, 31:60–76, 2019.
- 1642 27 Simon Bauer, Marcus Mohr, Ulrich Rüde, Jens Weismüller, Markus Wittmann, and Barbara
 1643 Wohlmuth. A two-scale approach for efficient on-the-fly operator assembly in massively parallel
 1644 high performance multigrid codes. *Applied Numerical Mathematics*, 122:14–38, 2017.
- 1645 28 Leonardo Bautista-Gomez, Seiji Tsuboi, Dimitri Komatitsch, Franck Cappello, Naoya
 1646 Maruyama, and Satoshi Matsuoka. FTI: high performance fault tolerance interface for hybrid
 1647 systems. In *Proceedings of 2011 international conference for high performance computing,*
 1648 *networking, storage and analysis*, pages 1–32, 2011.
- 1649 29 D. El Baz, P. Spitéri, J.C. Miellou, and D. Gazen. Asynchronous iterative algorithms with
 1650 flexible communication for non linear network flow problems. *Journal of Parallel and Distributed*
 1651 *Computing*, 38:1–15, 1996.
- 1652 30 D. A. Beckingsale, J. Burmark, R. Hornung, H. Jones, W. Killian, A. J. Kunen, O. Pearce,
 1653 P. Robinson, B. S. Ryujin, and T. R. Scogland. RAJA: Portable performance for large-scale
 1654 scientific applications. In *2019 IEEE/ACM International Workshop on Performance, Portability*
 1655 *and Productivity in HPC (P3HPC)*, pages 71–81, Nov 2019. doi:10.1109/P3HPC49587.2019.
 1656 00012.
- 1657 31 T. Benacchio, L. Bonaventura, M. Altenbernd, C.D. Cantwell, P.D. Düben, M. Gillard,
 1658 L. Giraud, D. Göttsche, E. Raffin, K. Teranishi, and N. Wedi. Report on local data recov-
 1659 ery approaches suitable for weather and climate prediction. FET-HPC ESCAPE-2 project
 1660 deliverable, 2020. doi:10.2172/1607968.
- 1661 32 T. Benacchio, L. Bonaventura, M. Altenbernd, C.D. Cantwell, P.D. Düben, M. Gillard,
 1662 L. Giraud, D. Göttsche, E. Raffin, K. Teranishi, and N. Wedi. Resilience and fault-tolerance
 1663 in high-performance computing for numerical weather and climate prediction. *International*
 1664 *Journal of High Performance Computing Applications*, 2020. Under revision.

- 1665 33 Anne Benoit, Aurélien Cavelan, Franck Cappello, Padma Raghavan, Yves Robert, and
 1666 Hongyang Sun. Identifying the right replication level to detect and correct silent errors
 1667 at scale. In *FTXS '17: Proceedings of the 2017 Workshop on Fault-Tolerance for HPC at*
 1668 *Extreme Scale*, pages 31–38, 06 2017. doi:10.1145/3086157.3086162.
- 1669 34 Anne Benoit, Aurélien Cavelan, Yves Robert, and Hongyang Sun. Optimal resilience patterns
 1670 to cope with fail-stop and silent errors. In *2016 IEEE International Parallel and Distributed*
 1671 *Processing Symposium (IPDPS)*, pages 202–211. IEEE, 2016.
- 1672 35 Austin R. Benson, Sven Schmit, and Robert Schreiber. Silent error detection in numerical
 1673 time-stepping schemes. *IJHPCA*, 29(4):403–421, 2015. arXiv:[https://doi.org/10.1177/](https://doi.org/10.1177/1094342014532297)
 1674 [1094342014532297](https://doi.org/10.1177/1094342014532297), doi:10.1177/1094342014532297.
- 1675 36 Marsha J. Berger, David L. George, Randall J. LeVeque, and Kyle T. Mandli. The GeoClaw
 1676 software for depth-averaged flows with adaptive refinement. *Advances in Water Resources*,
 1677 34(9):1195–1206, 2011.
- 1678 37 Eduardo Berrocal, Leonardo Bautista-Gomez, Sheng Di, Zhiling Lan, and Franck Cappello.
 1679 Exploring partial replication to improve lightweight silent data corruption detection for HPC
 1680 applications. In *Euro-Par 2016: Parallel Processing - 22nd International Conference on*
 1681 *Parallel and Distributed Computing, Grenoble, France, August 24-26, 2016, Proceedings*, pages
 1682 419–430, 2016. doi:10.1007/978-3-319-43659-3_31.
- 1683 38 Eduardo Berrocal, Leonardo Bautista-Gomez, Sheng Di, Zhiling Lan, and Franck Cappello.
 1684 Toward general software level silent data corruption detection for parallel applications. *IEEE*
 1685 *Trans. Parallel Distrib. Syst.*, 28(12):3642–3655, 2017. doi:10.1109/TPDS.2017.2735971.
- 1686 39 D. Bertsekas. Distributed asynchronous computation of fixed points. *Math. Programming*,
 1687 27:107 – 120, 1983.
- 1688 40 D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation, Numerical Methods*.
 1689 Prentice Hall, Englewood Cliffs N.J., 1989.
- 1690 41 Wesley Bland, Aurelien Bouteiller, Thomas Herault, Joshua Hursey, George Bosilca, and Jack J.
 1691 Dongarra. An evaluation of user-level failure mitigation support in MPI. In *Proceedings of the*
 1692 *19th European Conference on Recent Advances in the Message Passing Interface, EuroMPI'12*,
 1693 pages 193–203, Berlin, Heidelberg, 2012. Springer-Verlag. doi:10.1007/978-3-642-33518-1_
 1694 24.
- 1695 42 The OpenMP Architecture Review Boards. OpenMP (Open Multi-Processing), 2019. URL:
 1696 <https://www.openmp.org/>.
- 1697 43 L. Bonaventura, F. Casella, L. Delpopolo Carciopolo, and A. Ranade. A self adjusting
 1698 multirate algorithm for robust time discretization of partial differential equations. *Computers*
 1699 *and Mathematics with Applications*, 79:2086–2098, 2020.
- 1700 44 G. Bosilca, R. Delmas, J. Dongarra, and J. Langou. Algorithm-based fault tolerance applied
 1701 to high performance computing. *Journal of Parallel and Distributed Computing*, 69(4):410 –
 1702 416, 2009.
- 1703 45 Patrick G. Bridges, Kurt B. Ferreira, Michael A. Heroux, and Mark Hoemmen. Fault-tolerant
 1704 linear solvers via selective reliability, 2012. arXiv:1206.1390.
- 1705 46 S. Buchwald, Manuel Mohr, and Andreas Zwinkau. Malleable invasive applications. *CEUR*
 1706 *Workshop Proceedings*, 1337:123–126, 01 2015.
- 1707 47 H.-J. Bungartz and S. Dirnstorfer. Multivariate quadrature on adaptive sparse grids. *Computing*,
 1708 71(1):89–114, Aug 2003. doi:10.1007/s00607-003-0016-4.
- 1709 48 Hans-Joachim Bungartz and Michael Griebel. Sparse grids. *Acta Numerica*, 13:147–269, 2004.
 1710 doi:10.1017/S0962492904000182.
- 1711 49 Jon Calhoun, Luke Olson, and Marc Snir. Flipit: An llvm based fault injector for hpc. In
 1712 *Revised Selected Papers, Part I, of the Euro-Par 2014 International Workshops on Parallel*
 1713 *Processing - Volume 8805*, page 547–558, Berlin, Heidelberg, 2014. Springer-Verlag. doi:
 1714 [10.1007/978-3-319-14325-5_47](https://doi.org/10.1007/978-3-319-14325-5_47).

- 1715 50 Jon Calhoun, Luke Olson, Marc Snir, and William D. Gropp. Towards a more fault resilient
1716 multigrid solver. In *Proceedings of the Symposium on High Performance Computing*, San
1717 Diego, CA, USA, 2015. Society for Computer Simulation International.
- 1718 51 C. D. Cantwell and A. S. Nielsen. A minimally intrusive low-memory approach to resilience
1719 for existing transient solvers. *Journal of Scientific Computing*, 78(1):565–581, 2019.
- 1720 52 Franck Cappello, Al Geist, Bill Gropp, Laxmikant Kale, Bill Kramer, and Marc Snir. Toward
1721 exascale resilience. *The International Journal of High Performance Computing Applications*,
1722 23(4):374–388, 2009.
- 1723 53 L. Delpopolo Carciopolo, L. Bonaventura, A. Scotti, and L. Formaggia. A conservative implicit
1724 multirate method for hyperbolic problems. *Computational Geosciences*, 23:647–664, 2019.
- 1725 54 Ricolindo L. Carino, Ali Mohammed, and Florina M. Ciorba. Dynamic Loop Self-scheduling
1726 For Load Balancing (DLS4LB), 2020. URL: <https://github.com/unibas-dmi-hpc/DLS4LB>.
- 1727 55 Henri Casanova, Yves Robert, Frédéric Vivien, and Dounia Zaidouni. On the impact of process
1728 replication on executions of large-scale parallel applications with coordinated checkpointing.
1729 *Future Generation Comp. Syst.*, 51:7–19, 2015.
- 1730 56 Marc Casas, Wilfried N. Gansterer, and Elias Wimmer. Resilient gossip-inspired all-reduce al-
1731 gorithms for high-performance computing: Potential, limitations, and open questions. *IJHPCA*,
1732 33(2), 2019. doi:10.1177/1094342018762531.
- 1733 57 A. Cavelan, R. M. Cabezón, and Florina M. Ciorba. Detection of silent data corruptions
1734 in smoothed particle hydrodynamics simulations. In *Proceedings of the 19th IEEE/ACM*
1735 *International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2019)*, May 2019.
- 1736 58 A. Cavelan and F. M. Ciorba. Algorithm-based fault tolerance for parallel stencil computa-
1737 tions. In *IEEE International Conference on Cluster Computing (Cluster 2019)*, Albuquerque,
1738 September 2019.
- 1739 59 M. Chau, T. Garcia, and P. Spiteri. Asynchronous Schwarz methods applied to constrained
1740 mechanical structures in grid environment. *Advances in Engineering Software*, 74:1 – 15, 2014.
- 1741 60 D. Chazan and W. Miranker. Chaotic relaxation. *Linear Algebra and its Applications*, 2(2):199 –
1742 222, 1969. URL: <http://www.sciencedirect.com/science/article/pii/0024379569900287>,
1743 doi:[https://doi.org/10.1016/0024-3795\(69\)90028-7](https://doi.org/10.1016/0024-3795(69)90028-7).
- 1744 61 S. Chen, G. Bronevetsky, M. Casas-Guix, and L. Peng. Comprehensive algorithmic resilience for
1745 numeric applications. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore,
1746 CA (United States), 2013.
- 1747 62 Z. Chen and J. Dongarra. Algorithm-based fault tolerance for fail-stop failures. *IEEE*
1748 *Transactions on Parallel and Distributed Systems*, 19(12):1628–1641, 2008.
- 1749 63 Zizhong Chen. Online-ABFT: An online algorithm based fault tolerance scheme for soft error
1750 detection in iterative methods. *SIGPLAN Not.*, 48(8), February 2013. doi:10.1145/2517327.
1751 2442533.
- 1752 64 Andrew A. Chien, Pavan Balaji, Nan Dun, Aiman Fang, Hajime Fujita, Kamil Iskra, Zachary A.
1753 Rubenstein, Ziming Zheng, Jeff R. Hammond, Ignacio Laguna, D. Richards, Anshu Dubey,
1754 Brian van Straalen, Mark Hoemmen, Michael A. Heroux, Keita Teranishi, and Andrew R.
1755 Siegel. Exploring versioned distributed arrays for resilience in scientific applications. *IJHPCA*,
1756 31(6):564–590, 2017. doi:10.1177/1094342016664796.
- 1757 65 Edmond Chow, Hartwig Anzt, and Jack J. Dongarra. Asynchronous iterative algorithm for
1758 computing incomplete factorizations on GPUs. In Julian M. Kunkel and Thomas Ludwig,
1759 editors, *High Performance Computing - 30th International Conference, ISC High Performance*
1760 *2015, Frankfurt, Germany, July 12-16, 2015, Proceedings*, volume 9137 of *Lecture Notes in*
1761 *Computer Science*, pages 1–16. Springer, 2015. doi:10.1007/978-3-319-20119-1_1.
- 1762 66 Edmond Chow and Aftab Patel. Fine-grained parallel incomplete LU factorization. *SIAM J.*
1763 *Scientific Computing*, 37(2), 2015. doi:10.1137/140968896.
- 1764 67 Cluster File Systems, Inc. Lustre: A scalable, high-performance file system. White paper, Avail-
1765 able at [https://cse.buffalo.edu/faculty/tkosar/cse710/papers/lustre-whitepaper](https://cse.buffalo.edu/faculty/tkosar/cse710/papers/lustre-whitepaper.pdf).
1766 pdf, 2007.

- 1767 68 Evan Coleman, Masha Sosonkina, and Edmond Chow. Fault tolerant variants of the fine-
1768 grained parallel incomplete LU factorization. In Lukás Polok, Masha Sosonkina, William I.
1769 Thacker, and Josef Weinbub, editors, *Proceedings of the 25th High Performance Computing*
1770 *Symposium, Virginia Beach, VA, USA, April 23 - 26, 2017*, pages 15:1–15:12. ACM, 2017.
1771 URL: <http://dl.acm.org/citation.cfm?id=3108111>.
- 1772 69 Camille Coti, Thomas Herault, Pierre Lemarinier, Laurence Pilard, Ala Rezmerita, Eric
1773 Rodriguezb, and Franck Cappello. Blocking vs. non-blocking coordinated checkpointing for
1774 large-scale fault tolerant MPI. In *SC'06: Proceedings of the 2006 ACM/IEEE conference on*
1775 *Supercomputing*, pages 18–18. IEEE, 2006.
- 1776 70 S. P. Crago, D. I. Kang, M. Kang, R. Kost, K. Singh, J. Suh, and J. P. Walters. Programming
1777 models and development software for a space-based many-core processor. In *4th Int. Conf. on*
1778 *Space Mission Challenges for Information Technology*, pages 95–102. IEEE, 2011.
- 1779 71 C. Denis, P. de Oliveira Castro, and E. Petit. Verificarlo: checking floating point accuracy
1780 through Monte Carlo arithmetic. In *ARITH'23, Silicon Valley, USA*, Jul 2016.
- 1781 72 S. Di and F. Cappello. Fast error-bounded lossy HPC data compression with SZ. In *2016*
1782 *IEEE IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages
1783 730–739. IEEE, 2016.
- 1784 73 Sheng Di, Mohamed Slim Bouguerra, Leonardo Bautista-Gomez, and Franck Cappello. Opti-
1785 mization of multi-level checkpoint model for large scale HPC applications. In *2014 IEEE 28th*
1786 *International Parallel and Distributed Processing Symposium*, pages 1181–1190. IEEE, 2014.
- 1787 74 Sheng Di and Franck Cappello. Fast error-bounded lossy HPC data compression with SZ.
1788 In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages
1789 730–739. IEEE, 2016.
- 1790 75 Catello Di Martino, Zbigniew Kalbarczyk, Ravishankar K Iyer, Fabio Baccanico, Joseph Fullop,
1791 and William Kramer. Lessons learned from the analysis of system failures at petascale: The
1792 case of blue waters. In *2014 44th Annual IEEE/IFIP International Conference on Dependable*
1793 *Systems and Networks*, pages 610–621. IEEE, 2014. doi:10.1109/DSN.2014.62.
- 1794 76 Jose Monsalve Diaz, Swaroop Pophale, Kyle Friedline, Oscar Hernandez, David E Bernholdt,
1795 and Sunita Chandrasekaran. Evaluating support for openmp offload features. In *Proceedings*
1796 *of the 47th International Conference on Parallel Processing Companion*, pages 1–10, 2018.
- 1797 77 Edsger W Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications*
1798 *of the ACM*, 17(11):643–644, 1974.
- 1799 78 Daniel Drzisga, Lorenz John, Ulrich Rüde, Barbara Wohlmuth, and Walter Zulehner. On the
1800 analysis of block smoothers for saddle point problems. *SIAM Journal on Matrix Analysis*
1801 *and Applications*, 39(2):932–960, 2018. arXiv:<https://doi.org/10.1137/16M1106304>, doi:
1802 10.1137/16M1106304.
- 1803 79 P. Du, A. Bouteiller, G. Bosilca, T. Herault, and J. Dongarra. Algorithm-based fault tolerance
1804 for dense matrix factorizations. *ACM SIGPLAN notices*, 47(8):225–234, 2012.
- 1805 80 Alejandro Duran, Eduard Ayguadé, Rosa M. Badia, Jesús Labarta, Luis Martinell, Xavier
1806 Martorell, and Judit Planas. OmpSs: a proposal for programming heterogeneous multi-core
1807 architectures. *Parallel Processing Letters*, 21(2):173–193, 2011. URL: [http://dblp.uni-trier.](http://dblp.uni-trier.de/db/journals/ppl/pp121.html#DuranABLMMP11)
1808 [de/db/journals/ppl/pp121.html#DuranABLMMP11](http://dblp.uni-trier.de/db/journals/ppl/pp121.html#DuranABLMMP11).
- 1809 81 P. Eberhart, J. Brajard, P. Fortin, and F. Jézéquel. High performance numerical validation
1810 using stochastic arithmetic. *Reliable Computing*, 21:35–52, 2015.
- 1811 82 H. Carter Edwards, Christian R. Trott, and Daniel Sunderland. Kokkos: Enabling many-
1812 core performance portability through polymorphic memory access patterns. *Journal of*
1813 *Parallel and Distributed Computing*, 74(12):3202 – 3216, 2014. Domain-Specific Lan-
1814 guages and High-Level Frameworks for High-Performance Computing. URL: [http://www.](http://www.sciencedirect.com/science/article/pii/S0743731514001257)
1815 [sciencedirect.com/science/article/pii/S0743731514001257](http://www.sciencedirect.com/science/article/pii/S0743731514001257), doi:[https://doi.org/10.](https://doi.org/10.1016/j.jpdc.2014.07.003)
1816 [1016/j.jpdc.2014.07.003](https://doi.org/10.1016/j.jpdc.2014.07.003).

- 1817 83 Mireille El Haddad, José C. Garay, Frédéric Magoulès, and Daniel B. Szyld. Synchronous
1818 and asynchronous optimized Schwarz methods for one-way subdivision of bounded domains.
1819 *Numerical Linear Algebra and Applications*, 27:e2279, 2020. 30 pages.
- 1820 84 J. Elliott, M. Hoemmen, and F. Mueller. Evaluating the impact of SDC on the GMRES iterative
1821 solver. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages
1822 1193–1202, May 2014. doi:10.1109/IPDPS.2014.123.
- 1823 85 J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann. Combining
1824 partial redundancy and checkpointing for HPC. In *2012 IEEE 32nd International Conference
1825 on Distributed Computing Systems (ICDCS)*, pages 615–626, June 2012. doi:10.1109/ICDCS.
1826 2012.56.
- 1827 86 James Elliott. *Resilient Iterative Linear Solvers Running Through Errors*. PhD thesis, North
1828 Carolina State University, 2015.
- 1829 87 Christian Engelmann and Thomas Naughton. Toward a performance/resilience tool for hard-
1830 ware/software co-design of high-performance computing systems. In *2013 42nd International
1831 Conference on Parallel Processing*, pages 960–969. IEEE, 2013.
- 1832 88 Christian Engelmann, Geoffroy R. Vallée, Thomas Naughton, and Stephen L. Scott. Proactive
1833 fault tolerance using preemptive migration. In *Proceedings of the 17th Euromicro International
1834 Conference on Parallel, Distributed, and network-based Processing (PDP) 2009*, pages 252–257,
1835 Weimar, Germany, February 18-20, 2009. IEEE Computer Society, Los Alamitos, CA, USA.
1836 doi:10.1109/PDP.2009.31.
- 1837 89 Christian Engelmann, Geoffroy R. Vallée, and Swaroop Pophale. Concepts for OpenMP target
1838 offload resilience. In *Lecture Notes in Computer Science: Proceedings of the 15th International
1839 Workshop on OpenMP (IWOMP) 2019*, volume 11718, pages 78–93, Auckland, New Zealand,
1840 September 11-13, 2019. Springer Verlag, Berlin, Germany. doi:10.1007/978-3-030-28596-8_
1841 6.
- 1842 90 Charbel Farhat and Francois-Xavier Roux. A method of finite element tearing and intercon-
1843 necting and its parallel solution algorithm. *International Journal for Numerical Methods in
1844 Engineering*, 32(6):1205–1227, 1991.
- 1845 91 David Fiala, Frank Mueller, Christian Engelmann, Kurt Ferreira, Ron Brightwell, and Rolf
1846 Riesen. Detection and correction of silent data corruption for large-scale high-performance com-
1847 puting. In *Proceedings of the 25th IEEE/ACM International Conference on High Performance
1848 Computing, Networking, Storage and Analysis (SC) 2012*, pages 78:1–78:12, Salt Lake City, UT,
1849 USA, November 10-16, 2012. ACM Press, New York, NY, USA. doi:10.1109/SC.2012.49.
- 1850 92 David Fiala, Frank Mueller, Kurt Ferreira, and Christian Engelmann. Mini-Ckpts: Surviving
1851 OS failures in persistent memory. In *Proceedings of the 30th ACM International Conference
1852 on Supercomputing (ICS) 2016*, pages 7:1–7:14, Istanbul, Turkey, June 1-3, 2016. ACM Press,
1853 New York, NY, USA. doi:10.1145/2925426.2926295.
- 1854 93 David Fiala, Frank Mueller, and Kurt B. Ferreira. Flipsphere: A software-based DRAM error
1855 detection and correction library for HPC. In *Proceedings of the 20th IEEE/ACM International
1856 Symposium on Distributed Simulation and Real Time Applications (DS-RT) 2016*, pages
1857 19–28, London, UK, September 21-23, 2016. IEEE Computer Society, Los Alamitos, CA, USA.
1858 doi:10.1109/DS-RT.2016.27.
- 1859 94 P.K. Fok. A linearly fourth order multirate Runge–Kutta method with error control. *Journal
1860 of Scientific Computing*, pages 1–19, 2015.
- 1861 95 MPI Forum. The Message Passing Interface standard, 2019. URL: <https://www.mpi-forum.org/>.
- 1862 96 Michael Frechtling and Philip H. W. Leong. MCALIB: measuring sensitivity to rounding error
1863 with monte carlo programming. *ACM TOPLAS*, 37(2):1–25, 2015.
- 1864 97 Andreas Frommer and Daniel B. Szyld. On asynchronous iterations. *Journal of Computational
1865 and Applied Mathematics*, 123:201–216, 2000.
- 1866 98 François Févotte and Bruno Lathuilière. Debugging and optimization of HPC programs in
1867 mixed precision with the verrou tool. In *Computational Reproducibility at Exascale Workshop*
1868

- (CRE2018), in conjunction with the International Conference on High Performance Computing, Networking, Storage and Analysis (SC18), Dallas, USA, 2018.
- 99 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, Nov 2014. doi:10.1109/SC.2014.78.
- 100 Marc Gamell, Daniel S. Katz, Keita Teranishi, Michael A. Heroux, Rob F. Van der Wijngaart, Timothy G. Mattson, and Manish Parashar. Evaluating online global recovery with Fenix using application-aware in-memory checkpointing techniques. In *ICPP Workshops*, pages 346–355. IEEE Computer Society, 2016. URL: <http://dblp.uni-trier.de/db/conf/icppw/icppw2016.html#GamellKTHWMP16>.
- 101 Wilfried N. Gansterer, Gerhard Niederbrucker, Hana Straková, and Stefan Schulze Grotthoff. Robust distributed orthogonalization based on randomized aggregation. In Vassil N. Alexandrov, Al Geist, and Jack J. Dongarra, editors, *Proceedings of the second workshop on Scalable algorithms for large-scale systems, ScalA@SC 2011, Seattle, WA, USA, November 14, 2011*, pages 7–10. ACM, 2011. doi:10.1145/2133173.2133177.
- 102 Wilfried N. Gansterer, Gerhard Niederbrucker, Hana Straková, and Stefan Schulze Grotthoff. Scalable and fault tolerant orthogonalization based on randomized distributed data aggregation. *J. Comput. Sci.*, 4(6):480–488, 2013. doi:10.1016/j.jocs.2013.01.006.
- 103 José C. Garay, Frédéric Magoulès, and Daniel B. Szyld. Synchronous and asynchronous optimized Schwarz method for Poisson’s equation in rectangular domains. Technical Report 17-10-18, Department of Mathematics, Temple University, October 2017. Revised April 2018.
- 104 Jochen Garcke. Regression with the optimised combination technique. In *Proceedings of the 23rd international conference on Machine learning*, pages 321–328. ACM Press, 2006.
- 105 Jochen Garcke. A dimension adaptive sparse grid combination technique for machine learning. *ANZIAM Journal*, 48:725–740, 2007.
- 106 C.W. Gear and D.R. Wells. Multirate linear multistep methods. *BIT*, 24:484–502, 1984.
- 107 Giorgis Georgakoudis, Ignacio Laguna, Dimitrios S. Nikolopoulos, and Martin Schulz. Refine: Realistic fault injection via compiler-based instrumentation for accuracy, portability and speed. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '17*, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3126908.3126972.
- 108 Cijo George and Sathish S. Vadhiyar. ADFT: An adaptive framework for fault tolerance on large scale systems using application malleability. *Procedia Computer Science*, 9:166 – 175, 2012.
- 109 T. Gerstner and M. Griebel. Dimension-adaptive tensor-product quadrature. *Computing*, 71(1):65–87, Aug 2003. doi:10.1007/s00607-003-0015-5.
- 110 Thomas Gerstner and Michael Griebel. Numerical integration using sparse grids. *Numerical Algorithms*, 18(3):209, Jan 1998. doi:10.1023/A:1019129717644.
- 111 Roberto Gioiosa, Jose Carlos Sancho, Song Jiang, and Fabrizio Petrini. Transparent, incremental checkpointing at kernel level: a foundation for fault tolerance for parallel computers. In *SC'05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, pages 9–9. IEEE, 2005.
- 112 Christian Glusa, Erik G. Boman, Edmond Chow, Sivasankaran Rajamanickam, and Daniel B. Szyld. Scalable asynchronous domain decomposition solvers. Technical Report 19-10-11, Department of Mathematics, Temple University, October 2019.
- 113 Christian Glusa, Paritosh Ramanan, Erik G. Boman, Edmond Chow, and Sivasankaran Rajamanickam. Asynchronous one-level and two-level domain decomposition solvers. *CoRR*, abs/1808.08172, 2018. URL: <http://arxiv.org/abs/1808.08172>, arXiv:1808.08172.
- 114 Björn Gmeiner, Ulrich Rüde, Holger Stengel, Christian Waluga, and Barbara Wohlmuth. Towards textbook efficiency for parallel multigrid. *Numerical Mathematics: Theory, Methods and Applications*, 8(1):22–46, 2015.

- 115 Dominik Göddeke, Mirco Altenbernd, and Dirk Ribbrock. Fault-tolerant finite-element multigrid algorithms with hierarchically compressed asynchronous checkpointing. *Parallel Computing*, 49:117–135, October 2015. doi:10.1016/j.parco.2015.07.003.
- 116 Michael Griebel and Peter Oswald. Stochastic subspace correction methods and fault tolerance. *Mathematics of Computation*, 89(321):279–312, 2020.
- 117 Michael Griebel, Michael Schneider, and Christoph Zenger. A combination technique for the solution of sparse grid problems. In *Iterative Methods in Lin. Alg.*, pages 263–281, 1992.
- 118 The Khronos Group. OpenCL (Open Computing Language), 2020. URL: <https://www.khronos.org/opencv1/>.
- 119 Ray Grout, Hemanth Kolla, Michael Minion, and John Bell. Achieving algorithmic resilience for temporal integration through spectral deferred corrections. *Communications in Applied Mathematics and Computational Science*, 12(1):25–50, 2017.
- 120 Q. Guan, N. Debardeleben, S. Blanchard, and S. Fu. F-SEFI: A fine-grained soft error fault injection tool for profiling application vulnerability. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 1245–1254, May 2014. doi:10.1109/IPDPS.2014.128.
- 121 Pierre-Louis Guhur, Hong Zhang, Tom Peterka, Emil Constantinescu, and Franck Cappello. Lightweight and accurate silent data corruption detection in ordinary differential equation solvers. In *European Conference on Parallel Processing*, pages 644–656. Springer, 2016.
- 122 E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer-Verlag, Berlin Heidelberg, 3rd corr. edition, 2008.
- 123 Brendan Harding et al. Fault tolerant computation with the sparse grid combination technique. *SIAM Journal on Scient. Comp.*, 37(3):C331–C353, 2015.
- 124 Brendan Harding and Markus Hegland. Robust solutions to PDEs with multiple grids. In Jochen Garcke and Dirk Pflüger, editors, *Sparse Grids and Applications - Munich 2012 SE*, volume 97 of *Lecture Notes in Computational Science and Engineering*, pages 171–193. Springer International Publishing, 2014.
- 125 Paul H. Hargrove and Jason C. Duell. Berkeley Lab Checkpoint/Restart (BLCR) for Linux clusters. In *Journal of Physics: Proceedings of the Scientific Discovery through Advanced Computing Program (SciDAC) Conference 2006*, volume 46, pages 494–499, Denver, CO, USA, June 25–29, 2006. Institute of Physics Publishing, Bristol, UK. doi:10.1088/1742-6596/46/1/067.
- 126 Siva Kumar Sastry Hari, Timothy Tsai, Mark Stephenson, Stephen W. Keckler, and Joel S. Emer. SASSIFI: an architecture-level fault injection tool for GPU application resilience evaluation. In *2017 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2017, Santa Rosa, CA, USA, April 24–25, 2017*, pages 249–258. IEEE Computer Society, 2017. doi:10.1109/ISPASS.2017.7975296.
- 127 Amin Hassani, Anthony Skjellum, and Ron Brightwell. Design and evaluation of FA-MPI, a transactional resilience scheme for non-blocking MPI. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23–26, 2014*, pages 750–755. IEEE Computer Society, 2014. doi:10.1109/DSN.2014.78.
- 128 Xubin (Ben) He, Li Ou, Christian Engelmann, Xin Chen, and Stephen L. Scott. Symmetric active/active metadata service for high availability parallel file systems. *Journal of Parallel and Distributed Computing (JPDC)*, 69(12):961–973, December 2009. doi:10.1016/j.jpdc.2009.08.004.
- 129 Mario Heene, Alfredo Parra Hinojosa, Michael Obersteiner, Hans-Joachim Bungartz, and Dirk Pflüger. EXAHD: An exa-scalable two-level sparse grid approach for higher-dimensional problems in plasma physics and beyond. In *High Performance Computing in Science and Engineering'17*, pages 513–529. Springer, 2018.
- 130 Mario Heene, Alfredo Parra Hinojosa, Hans-Joachim Bungartz, and Dirk Pflüger. A massively-parallel, fault-tolerant solver for high-dimensional PDEs. In *Euro-Par 2016: Parallel Processing Workshops*, 2016.

- 1973 131 Thomas Herault and Yves Robert, editors. *Fault-Tolerance Techniques for High-Performance*
1974 *Computing*. Springer Verlag, 2015.
- 1975 132 Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear
1976 systems. *Journal of Research of the National Bureau of Standards*, 46(6):409–436, December
1977 1952.
- 1978 133 N.J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and
1979 Applied Mathematics, Philadelphia, PA, USA, 1st edition, 1996.
- 1980 134 Alfredo Parra Hinojosa, Brendan Harding, Markus Hegland, and Hans-Joachim Bungartz.
1981 Handling silent data corruption with the sparse grid combination technique. In *Software for*
1982 *Exascale Computing-SPPEXA 2013-2015*, pages 187–208. Springer, 2016.
- 1983 135 Mark Hoemmen and Michael Allen Heroux. Fault-tolerant iterative methods via selec-
1984 tive reliability. *Proceedings of the IEEE/ACM conference on supercomputing (SC’11)*,
1985 2011. URL: [http://www.researchgate.net/publication/228940928_Fault-Tolerant_](http://www.researchgate.net/publication/228940928_Fault-Tolerant_Iterative_Methods_via_Selective_Reliability/file/79e41508060305e4ad.pdf)
1986 [Iterative_Methods_via_Selective_Reliability/file/79e41508060305e4ad.pdf](http://www.researchgate.net/publication/228940928_Fault-Tolerant_Iterative_Methods_via_Selective_Reliability/file/79e41508060305e4ad.pdf).
- 1987 136 K.-H. Huang and J.A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE*
1988 *Transactions on Computers*, 100(6):518–528, 1984.
- 1989 137 Kuang-hua Huang and Jacob a Abraham. Algorithm-based fault tolerance for matrix operations.
1990 *IEEE Transactions on Computers*, c(6):518–528, 1984. doi:10.1109/TC.1984.1676475.
- 1991 138 Markus Huber, Björn Gmeiner, Ulrich Rüde, and Barbara Wohlmuth. Resilience for massively
1992 parallel multigrid solvers. *SIAM Journal on Scientific Computing*, 38(5):S217–S239, 2016.
- 1993 139 Markus Huber, Ulrich Rüde, and Barbara Wohlmuth. Adaptive control in roll-forward recovery
1994 for extreme scale multigrid. *The International Journal of High Performance Computing*
1995 *Applications*, 33(5):817–837, 2019.
- 1996 140 S. Hukerikar, K. Teranishi, P. C. Diniz, and R. F. Lucas. An evaluation of lazy fault detection
1997 based on adaptive redundant multithreading. In *2014 IEEE High Performance Extreme*
1998 *Computing Conference (HPEC)*, pages 1–6, 2014.
- 1999 141 Saurabh Hukerikar and Christian Engelmann. A pattern language for high-performance
2000 computing resilience. In *Proceedings of the 22nd European Conference on Pattern Languages*
2001 *of Programs (EuroPLoP) 2017*, pages 12:1–12:16, Kloster Irsee, Germany, July 12-16, 2017.
2002 ACM Press, New York, NY, USA. doi:10.1145/3147704.3147718.
- 2003 142 Saurabh Hukerikar and Christian Engelmann. Resilience design patterns: A structured
2004 approach to resilience at extreme scale (version 1.2). Technical Report ORNL/TM-2017/745,
2005 Oak Ridge National Laboratory, Oak Ridge, TN, USA, August 2017. doi:10.2172/1436045.
- 2006 143 Frank Hülsemann, Markus Kowarschik, Marcus Mohr, and Ulrich Rüde. Parallel geometric
2007 multigrid. In *Numerical Solution of Partial Differential Equations on Parallel Computers*,
2008 pages 165–208. Springer, 2006.
- 2009 144 R. K. Iyer, N. M. Nakka, Z. T. Kalbarczyk, and S. Mitra. Recent advances and new
2010 avenues in hardware-level reliability support. *IEEE Micro*, 25(6):18–29, Nov 2005. doi:
2011 10.1109/MM.2005.119.
- 2012 145 John D. Jakeman and Stephen G. Roberts. Local and dimension adaptive sparse grid
2013 interpolation and quadrature. *arXiv preprint arXiv:1110.0010*, September, 2011. arXiv:
2014 1110.0010v1.
- 2015 146 L. Jaulmes, M. Casas, M. Moretó ans E. Ayguadé, J. Labarta, and M. Valero. Exploiting
2016 asynchrony from exact forward recovery for detected and uncorrected errors in iterative solvers.
2017 In *SC ’15: Proceedings of the International Conference for High Performance Computing,*
2018 *Networking, Storage and Analysis*, Austin, TX, 2015.
- 2019 147 Y. Jia, G. Bosilca, P. Luszczek, and J. J. Dongarra. Parallel reduction to Hessenberg form
2020 with algorithm-based fault tolerance. In *Proceedings of the International Conference on High*
2021 *Performance Computing, Networking, Storage and Analysis*, SC ’13, New York, NY, USA,
2022 2013. Association for Computing Machinery. doi:10.1145/2503210.2503249.

- 2023 148 A. Jin, J. Jiang, J. Hu, and J. Lou. A pin-based dynamic software fault injection system. In
2024 2008 *The 9th International Conference for Young Computer Scientists*, pages 2160–2167, Nov
2025 2008. doi:10.1109/ICYCS.2008.329.
- 2026 149 U. Kabir and D. Goswami. An ABFT scheme based on communication characteristics. In
2027 2016 *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 515–523, Sep.
2028 2016. doi:10.1109/CLUSTER.2016.68.
- 2029 150 Hartmut Kaiser, Thomas Heller, Bryce Adelstein-Lelbach, Adrian Serio, and Dietmar Fey.
2030 HPX: A task based programming model in a global address space. In *Proceedings of the 8th*
2031 *International Conference on Partitioned Global Address Space Programming Models*, PGAS'14,
2032 New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2676870.
2033 2676883.
- 2034 151 Laxmikant V. Kale and Sanjeev Krishnan. Charm++: A portable concurrent object oriented
2035 system based on C++. In *Proceedings of the Eighth Annual Conference on Object-Oriented*
2036 *Programming Systems, Languages, and Applications*, OOPSLA'93, page 91–108, New York,
2037 NY, USA, 1993. Association for Computing Machinery. doi:10.1145/165854.165874.
- 2038 152 George Karniadakis and Spencer Sherwin. *Spectral/hp element methods for computational*
2039 *fluid dynamics*. Oxford University Press, 2013.
- 2040 153 Kai Keller and Leonardo Bautista-Gomez. Application-level differential checkpointing for HPC
2041 applications with dynamic datasets. In *19th IEEE/ACM International Symposium on Cluster,*
2042 *Cloud and Grid Computing, CCGRID 2019, Larnaca, Cyprus, May 14-17, 2019*, pages 52–61.
2043 IEEE, 2019. doi:10.1109/CCGRID.2019.00015.
- 2044 154 Jungwon Kim, Seyong Lee, and Jeffrey S. Vetter. PapyrusKV: A high-performance parallel key-
2045 value store for distributed NVM architectures. In *Proceedings of the International Conference*
2046 *for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA,
2047 2017. Association for Computing Machinery. doi:10.1145/3126908.3126943.
- 2048 155 Axel Klawonn, Martin J. Kühn, and Oliver Rheinbach. Parallel adaptive FETI-DP using
2049 lightweight asynchronous dynamic load balancing. *International Journal for Numerical*
2050 *Methods in Engineering*, 121(4):621–643, 2020. URL: [https://onlinelibrary.wiley.com/](https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.6237)
2051 [doi/pdf/10.1002/](https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.6237)
2052 [nme.6237](https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.6237), doi:10.1002/nme.6237.
- 2053 156 Nils Kohl, Johannes Hötzer, Florian Schornbaum, Martin Bauer, Christian Godenschwager,
2054 Harald Köstler, Britta Nestler, and Ulrich Rüde. A scalable and extensible checkpointing
2055 scheme for massively parallel simulations. *The International Journal of High Performance*
2056 *Computing Applications*, 33(4):571–589, 2019.
- 2057 157 Giorgos Kollias, Efstratios Gallopoulos, and Daniel B. Szyld. Asynchronous iterative com-
2058 putations with Web information retrieval structures: The PageRank case. In G.R. Joubert,
2059 W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, and E. Zapata, editors, *Parallel Computing:*
2060 *Current and Future Issues of High-End Computing (Proceedings of the International Confer-*
2061 *ence Parco05)*, volume 33 of *NIC Series*, pages 309–316, Jülich, Germany, 2006. John von
2062 Neumann-Institut für Computing (NIC).
- 2063 158 U.W. Kulisch. *Advanced Arithmetic for the Digital Computer*. Springer, Wien, 2002.
- 2064 159 Ignacio Laguna, David Richards, Todd Gamblin, Martin Schulz, Bronis Supinski, Kathryn
2065 Mohror, and Howard Pritchard. Evaluating and extending user-level fault tolerance in MPI
2066 applications. *The International Journal of High Performance Computing Applications*, 30, 01
2067 2016. doi:10.1177/1094342015623623.
- 2068 160 J.D. Lambert. *Numerical Methods for Ordinary Differential Systems*. Wiley, Chirchester,
2069 England, 1991.
- 2070 161 Julien Langou, Zizhong Chen, George Bosilca, and Jack Dongarra. Recovery patterns for
2071 iterative methods in a parallel unstable environment. *SIAM J. Sci. Comput.*, 30:102–116,
2072 November 2007. doi:10.1137/040620394.
- 2073 162 Christian Lengauer, Sven Apel, Mathias Bolten, Shigeru Chiba, Ulrich Rüde, Jürgen Teich,
2074 Armin Größlinger, Frank Hannig, Harald Köstler, Lisa Claus, Alexander Grebhahn, Stefan

- Groth, Stefan Kronawitter, Sebastian Kuckuk, Hannah Rittich, Christian Schmitt, and Jonas Schmitt. ExaStencils – Advanced Multigrid Solver Generation. In Hans-Joachim Bungartz, Severin Reiz, Philipp Neumann, Benjamin Uekermann, and Wolfgang Nagel, editors, *Software for Exascale Computing – SPPEXA 2016-2019*, Lecture Notes in Computer Science and Engineering. Springer, 2020. URL: https://www12.cs.fau.de/downloads/hannig/publications/ExaStencils_Advanced_Multigrid_Solver_Generation.pdf.
- 163 Randall J. LeVeque, David L. George, and Marsha J. Berger. Tsunami modelling with adaptively refined finite volume methods. *Acta Numerica*, 20:211–289, 2011.
- 164 Markus Levonyak, Christina Pacher, and Wilfried N. Gansterer. Scalable resilience against node failures for communication-hiding preconditioned conjugate gradient and conjugate residual methods. In *Proceedings of the 2020 SIAM Conference on Parallel Processing for Scientific Computing*, pages 81–92. SIAM, 2020. URL: <https://epubs.siam.org/doi/abs/10.1137/1.9781611976137.8>, arXiv:<https://epubs.siam.org/doi/pdf/10.1137/1.9781611976137.8>, doi:10.1137/1.9781611976137.8.
- 165 Scott Levy, Bryan Topp, Kurt B Ferreira, Dorian Arnold, Torsten Hoefer, and Patrick Widener. Using simulation to evaluate the performance of resilience strategies at scale. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, pages 91–114. Springer, 2013.
- 166 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.
- 167 Xin Liang, Jieyang Chen, Dingwen Tao, Sihuan Li, Panruo Wu, Hongbo Li, Kaiming Ouyang, Yuanlai Liu, Fengguang Song, and Zizhong Chen. Correcting soft errors online in fast Fourier transform. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3126908.3126915.
- 168 J. Lidman, D. J. Quinlan, C. Liao, and S. A. McKee. ROSE::FTTransform - a source-to-source translation framework for exascale fault-tolerance research. In *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)*, pages 1–6, June 2012. doi:10.1109/DSNW.2012.6264672.
- 169 J. Liesen and Z. Strakoš. *Krylov Subspace Methods*. Numerical Mathematics and Scientific Computation. Oxford University Press, 2013.
- 170 S. Lin and P. Chen. A simd-based software fault tolerance for arm processors. In *2017 International Conference on Applied System Innovation (ICASI)*, pages 910–913, 2017.
- 171 Romain Lion. Tolérance aux pannes dans l’exécution distribuée de graphes de tâches. In *Conférence d’informatique en Parallélisme, Architecture et Système*, Anglet, France, June 2019. URL: <https://hal.inria.fr/hal-02296118>.
- 172 Romain Lion and Samuel Thibault. From tasks graphs to asynchronous distributed checkpointing with local restart. In *Fault Tolerance for HPC at eXtreme Scale (FTXS) Workshop*, 2020.
- 173 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.
- 174 Kaoutar Maghraoui, Travis Desell, Boleslaw Szymanski, and Carlos Varela. Dynamic malleability in iterative MPI applications. In *In: 7th International Symposium on Cluster Computing and the Grid*, pages 591–598, 05 2007. doi:10.1109/CCGRID.2007.45.
- 175 Frédéric Magoulès, Daniel B. Szyld, and Cédric Venet. Asynchronous optimized Schwarz methods with and without overlap. *Numerische Mathematik*, 137:199–227, 2017.
- 176 Tim Mattson, Romain Cledat, Vincent Cave, Vivek Sarkar, Zoran Budimlic, Sanjay Chatterjee, Josh Fryman, Ivan Ganev, Robin Knauerhase, Min Lee, Benoit Meister, Brian Nickerson, Nick Pepperling, Bala Seshasayee, Sagnak Tasirlar, Justin Teller, and Nick Vrvilo. The open community runtime: A runtime system for extreme scale computing. In *2016 IEEE High*

- 2127 *Performance Extreme Computing Conference (HPEC)*, pages 1–7, 09 2016. doi:10.1109/
2128 HPEC.2016.7761580.
- 2129 **177** Harshitha Menon and Kathryn Mohror. DisCVar: Discovering critical variables using
2130 algorithmic differentiation for transient faults. *SIGPLAN Not.*, 53(1), February 2018.
2131 doi:10.1145/3200691.3178502.
- 2132 **178** G. Meurant and Z. Strakoš. The Lanczos and Conjugate Gradient algorithms in finite precision
2133 arithmetic. *Acta Numerica*, 15:471–542, 2006.
- 2134 **179** Amitabh Mishra and Prithviraj Banerjee. An algorithm-based error detection scheme for
2135 the multigrid method. *IEEE Trans. Comput.*, 52(9), September 2003. doi:10.1109/TC.2003.
2136 1228507.
- 2137 **180** A. Mohammed, A. Cavelan, and F. M. Ciorba. rDLB: A novel approach for robust dynamic
2138 load balancing of scientific applications with independent tasks. In *Proceedings of the 2019*
2139 *International Conference on High Performance Computing & Simulation (HPCS 2019)*, Dublin,
2140 July 2019.
- 2141 **181** Michael Moldaschl, Karl E. Prikopa, and Wilfried N. Gansterer. Fault tolerant communication-
2142 optimal 2.5D matrix multiplication. *J. Parallel Distributed Comput.*, 104:179–190, 2017.
2143 doi:10.1016/j.jpdc.2017.01.022.
- 2144 **182** A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. Design, modeling, and
2145 evaluation of a scalable multi-level checkpointing system. In *SC '10: Proceedings of the 2010*
2146 *ACM/IEEE International Conference for High Performance Computing, Networking, Storage*
2147 *and Analysis*, pages 1–11, Nov 2010. doi:10.1109/SC.2010.18.
- 2148 **183** Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis R. de Supinski. Design,
2149 modeling, and evaluation of a scalable multi-level checkpointing system. In *Proceedings of the*
2150 *2010 ACM/IEEE International Conference for High Performance Computing, Networking,*
2151 *Storage and Analysis*, SC '10, page 1–11, USA, 2010. IEEE Computer Society. doi:10.1109/
2152 SC.2010.18.
- 2153 **184** K. Morris, F. Rizzi, B. Cook, P. Mycek, O. LeMaitre, O. M. Knio, K. Sargsyan, K. Dahlgren,
2154 and B. J. Debusschere. Performance scaling variability and energy analysis for a resilient
2155 ULFM-based PDE solver. In *2016 7th Workshop on Latest Advances in Scalable Algorithms*
2156 *for Large-Scale Systems (ScalA)*, pages 41–48, Nov 2016. doi:10.1109/ScalA.2016.010.
- 2157 **185** Andreas Müller, Jörn Behrens, Francis X Giraldo, and Volkmar Wirth. Comparison between
2158 adaptive and uniform discontinuous Galerkin simulations in dry 2D bubble experiments.
2159 *Journal of Computational Physics*, 235:371–393, 2013.
- 2160 **186** Paul Mycek, Francesco Rizzi, Olivier Le Maître, Khachik Sargsyan, Karla Morris, Cosmin Safta,
2161 Bert Debusschere, and Omar Knio. Discrete a priori bounds for the detection of corrupted PDE
2162 solutions in exascale computations. *SIAM Journal on Scientific Computing*, 39(1):C1–C28,
2163 2017. arXiv:https://doi.org/10.1137/15M1051786, doi:10.1137/15M1051786.
- 2164 **187** Arun B. Nagarajan, Frank Mueller, Christian Engelmann, and Stephen L. Scott. Proactive
2165 fault tolerance for HPC with Xen virtualization. In *Proceedings of the 21st ACM International*
2166 *Conference on Supercomputing (ICS) 2007*, pages 23–32, Seattle, WA, USA, June 16–20, 2007.
2167 ACM Press, New York, NY, USA. doi:10.1145/1274971.1274978.
- 2168 **188** X. Ni and L. V. Kale. FlipBack: Automatic targeted protection against silent data corruption.
2169 In *Conference for High Performance Computing, Networking, Storage and Analysis (SC)*,
2170 pages 335–346, Nov 2016. doi:10.1109/SC.2016.28.
- 2171 **189** Bogdan Nicolae, Adam Moody, Elsa Gonsiorowski, Kathryn Mohror, and Franck Cappello.
2172 VeloC: Towards high performance adaptive asynchronous checkpointing at large scale. In
2173 *IPDPS'19: The 2019 IEEE International Parallel and Distributed Processing Symposium*,
2174 pages 911–920, Rio de Janeiro, Brazil, May 2019. URL: https://hal.archives-ouvertes.
2175 fr/hal-02184203.
- 2176 **190** Gerhard Niederbrucker and Wilfried N. Gansterer. Robust gossip-based aggregation: A
2177 practical point of view. In Peter Sanders and Norbert Zeh, editors, *Proceedings of the 15th*

- 2178 *Meeting on Algorithm Engineering and Experiments, ALENEX 2013, New Orleans, Louisiana,*
 2179 *USA, January 7, 2013*, pages 133–147. SIAM, 2013. doi:10.1137/1.9781611972931.12.
- 2180 **191** Allan S. Nielsen and Jan S. Hesthaven. Fault tolerance in the Parareal method. In *Proceedings*
 2181 *of the ACM Workshop on Fault-Tolerance for HPC at Extreme Scale, FTXS '16*, pages 1–8,
 2182 New York, NY, USA, 2016. ACM. URL: <http://dx.doi.org/10.1145/2909428.2909431>,
 2183 doi:10.1145/2909428.2909431.
- 2184 **192** Michael Obersteiner, Alfredo Parra Hinojosa, Mario Heene, Hans-Joachim Bungartz, and
 2185 Dirk Pflüger. A highly scalable, algorithm-based fault-tolerant solver for gyrokinetic plasma
 2186 simulations. In *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms*
 2187 *for Large-Scale Systems*, pages 1–8, 2017.
- 2188 **193** N. Oh, P. P. Shirvani, and E. J. McCluskey. Error detection by duplicated instructions in
 2189 super-scalar processors. *IEEE Transactions on Reliability*, 51(1):63–75, March 2002. doi:
 2190 10.1109/24.994913.
- 2191 **194** C. Pachajoa, C. Pacher, and W. N. Gansterer. Node-failure-resistant preconditioned conjugate
 2192 gradient method without replacement nodes. In *2019 IEEE/ACM 9th Workshop on Fault*
 2193 *Tolerance for HPC at eXtreme Scale (FTXS)*, pages 31–40, Nov 2019. doi:10.1109/FTXS49593.
 2194 2019.00009.
- 2195 **195** Carlos Pachajoa and Wilfried N. Gansterer. On the resilience of conjugate gradient and
 2196 multigrid methods to node failures. In Dora Blanco Heras, Luc Bougé, Gabriele Mencagli,
 2197 Emmanuel Jeannot, Rizos Sakellariou, Rosa M. Badia, Jorge G. Barbosa, Laura Ricci,
 2198 Stephen L. Scott, Stefan Lankes, and Josef Weidendorfer, editors, *Euro-Par 2017: Parallel*
 2199 *Processing Workshops - Euro-Par 2017 International Workshops, Santiago de Compostela,*
 2200 *Spain, August 28-29, 2017, Revised Selected Papers*, volume 10659 of *Lecture Notes in Computer*
 2201 *Science*, pages 569–580. Springer, 2017. doi:10.1007/978-3-319-75178-8_46.
- 2202 **196** Carlos Pachajoa, Markus Levonyak, and Wilfried N. Gansterer. Extending and evaluating
 2203 fault-tolerant preconditioned conjugate gradient methods. In *IEEE/ACM 8th Workshop on*
 2204 *Fault Tolerance for HPC at eXtreme Scale, FTXS@SC 2018, Dallas, TX, USA, November 16,*
 2205 *2018*, pages 49–58. IEEE, 2018. doi:10.1109/FTXS.2018.00009.
- 2206 **197** Carlos Pachajoa, Markus Levonyak, Wilfried N. Gansterer, and Jesper Larsson Träff. How to
 2207 make the preconditioned conjugate gradient method resilient against multiple node failures.
 2208 In *Proceedings of the 48th International Conference on Parallel Processing, ICPP 2019, Kyoto,*
 2209 *Japan, August 05-08, 2019*, pages 67:1–67:10. ACM, 2019. doi:10.1145/3337821.3337849.
- 2210 **198** D. Stott Parker, Brad Pierce, and Paul R. Eggert. Monte Carlo arithmetic: a framework for
 2211 the statistical analysis of roundoff error. *IEEE Computation in Science and Engineering*, 2001.
- 2212 **199** Alfredo Parra Hinojosa, Christoph Kowitz, Mario Heene, Dirk Pflüger, and H-J Bungartz.
 2213 Towards a fault-tolerant, scalable implementation of GENE. In *Recent Trends in Computational*
 2214 *Engineering-CE2014*, pages 47–65. Springer, 2015.
- 2215 **200** Sri Raj Paul, Akihiro Hayashi, Nicole Slattengren, Hemanth Kolla, Matthew Whitlock,
 2216 Seonmyeong Bak, Keita Teranishi, Jackson Mayo, and Vivek Sarkar. Enabling resilience in
 2217 asynchronous many-task programming models. In Ramin Yahyapour, editor, *Euro-Par 2019:*
 2218 *Parallel Processing*, pages 346–360. Springer International Publishing, 2019.
- 2219 **201** Benjamin Peherstorfer, Dirk Pflüger, and Hans-Joachim Bungartz. Density estimation
 2220 with adaptive sparse grids for large data sets. In *Proceedings of the 2014 SIAM inter-*
 2221 *national conference on data mining*, pages 443–451. SIAM, 2014. URL: [https://epubs.siam.](https://epubs.siam.org/doi/abs/10.1137/1.9781611973440.51)
 2222 [org/doi/abs/10.1137/1.9781611973440.51](https://epubs.siam.org/doi/pdf/10.1137/1.9781611973440.51), arXiv:[https://epubs.siam.org/doi/pdf/10.](https://epubs.siam.org/doi/pdf/10.1137/1.9781611973440.51)
 2223 [1137/1.9781611973440.51](https://epubs.siam.org/doi/pdf/10.1137/1.9781611973440.51), doi:10.1137/1.9781611973440.51.
- 2224 **202** A. J. Pena, W. Bland, and P. Balaji. VOCL-FT: Introducing techniques for efficient soft error
 2225 coprocessor recovery. In *Proceedings of the International Conference on High Performance*
 2226 *Computing, Networking, Storage and Analysis (SC'15)*, pages 1–12, Nov 2015. doi:10.1145/
 2227 2807591.2807640.

- 2228 **203** M.D. Piggott, P.E. Farrell, C.R. Wilson, G.J. Gorman, and C.C. Pain. Anisotropic mesh
2229 adaptivity for multi-scale ocean modelling. *Philosophical Transactions of the Royal Society A:*
2230 *Mathematical, Physical and Engineering Sciences*, 367(1907):4591–4611, 2009.
- 2231 **204** A. Poulos, D. Wallace, R. Robey, L. Monroe, V. Job, S. Blanchard, W. Jones, and N. De-
2232 Bardeleben. Improving application resilience by extending error correction with contextual
2233 information. In *2018 IEEE/ACM 8th Workshop on Fault Tolerance for HPC at eXtreme Scale*
2234 *(FTXS)*, pages 19–28, Nov 2018. doi:10.1109/FTXS.2018.00006.
- 2235 **205** Karl E. Prikopa and Wilfried N. Gansterer. Fault-tolerant least squares solvers for wireless
2236 sensor networks based on gossiping. *J. Parallel Distributed Comput.*, 136:52–62, 2020. doi:
2237 10.1016/j.jpdc.2019.09.006.
- 2238 **206** Alfio Quarteroni and Alberto Valli. *Numerical approximation of partial differential equations*,
2239 volume 23. Springer Science & Business Media, 2008.
- 2240 **207** Petar Radojkovic, Manolis Marazakis, Paul Carpenter, Reiley Jeyapaul, Dimitris Gizopoulos,
2241 Martin Schulz, Adria Armejach, Eduard A Ayguade, François Bodin, Ramon Canal, Franck
2242 Cappello, Fabien Chaix, Guillaume Colin De Verdiere, Said Derradji, Stefano Di Carlo,
2243 Christian Engelmann, Ignacio Laguna, Miquel Moreto, Onur Mutlu, Lazaros Papadopoulos,
2244 Olly Perks, Manolis Ploumidis, Bezhad Salami, Yanos Sazeides, Dimitrios Soudris, Yiannis
2245 Soudris, Per Stenstrom, Samuel Thibault, Will Toms, and Osman Unsal. Towards Resilient
2246 EU HPC Systems: A Blueprint. Research report, European HPC resilience initiative, April
2247 2020. URL: <https://hal.inria.fr/hal-02922257>.
- 2248 **208** M. Wasiur Rashid and Michael C. Huang. Supporting highly-decoupled thread-level redundancy
2249 for parallel programs. In *Conf. on High-Performance Computer Architecture (HPCA)*, pages
2250 393–404. IEEE, 2008.
- 2251 **209** G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August. Swift: software imple-
2252 mented fault tolerance. In *International Symposium on Code Generation and Optimization*,
2253 pages 243–254, 2005.
- 2254 **210** F. Rizzi, K. Morris, K. Sargsyan, P. Mycek, C. Safta, O. Le Maître, O.M. Knio, and B.J. De-
2255 busschere. Exploring the interplay of resilience and energy consumption for a task-based partial
2256 differential equations preconditioner. *Parallel Computing*, 73:16 – 27, 2018. Parallel Program-
2257 ming for Resilience and Energy Efficiency. URL: [http://www.sciencedirect.com/science/](http://www.sciencedirect.com/science/article/pii/S0167819117300753)
2258 [article/pii/S0167819117300753](http://www.sciencedirect.com/science/article/pii/S0167819117300753), doi:<https://doi.org/10.1016/j.parco.2017.05.005>.
- 2259 **211** Francesco Rizzi, Karla Morris, Khachik Sargsyan, Paul Mycek, Cosmin Safta, Bert Debusschere,
2260 Olivier LeMaitre, and Omar Knio. ULFM-MPI implementation of a resilient task-based partial
2261 differential equations preconditioner. In *Proceedings of the ACM Workshop on Fault-Tolerance*
2262 *for HPC at Extreme Scale*, New York, NY, USA, 2016. Association for Computing Machinery.
2263 doi:10.1145/2909428.2909429.
- 2264 **212** Gabriel Rodríguez, María J. Martín, Patricia González, Juan Touriño, and Ramón Doallo.
2265 Cppc: A compiler-assisted tool for portable checkpointing of message-passing applications.
2266 *Concurr. Comput.: Pract. Exper.*, 22(6):749–766, April 2010.
- 2267 **213** T. Ropars, T. V. Martsinkevich, A. Guermouche, A. Schiper, and F. Cappello. SPBC:
2268 Leveraging the characteristics of MPI HPC applications for scalable checkpointing. In *SC '13:*
2269 *Proceedings of the International Conference on High Performance Computing, Networking,*
2270 *Storage and Analysis*, pages 1–12, Nov 2013. doi:10.1145/2503210.2503271.
- 2271 **214** Ulrich Rüde. Fully adaptive multigrid methods. *SIAM Journal on Numerical Analysis*,
2272 30(1):230–248, 1993.
- 2273 **215** Ulrich Rüde. *Mathematical and computational techniques for multilevel adaptive methods*.
2274 SIAM, 1993.
- 2275 **216** Ulrich Rüde. Error estimators based on stable splittings. In David E Keyes and Jinchao Xu,
2276 editors, *Domain Decomposition Methods in Scientific and Engineering Computing: Proceedings*
2277 *of the Seventh International Conference on Domain Decomposition, October 27-30, 1993, the*
2278 *Pennsylvania State University*, volume 180 of *Contemporary Mathematics*, pages 111–118.
2279 American Mathematical Soc., 1994.

- 2280 **217** Adrian Sandu. A class of multirate infinitesimal gark methods. *SIAM Journal on Numerical*
2281 *Analysis*, 57(5):2300–2327, 2019.
- 2282 **218** Piyush Sao and Richard Vuduc. Self-stabilizing iterative solvers. *Proceedings of the*
2283 *Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems - ScalA '13*,
2284 pages 1–8, 2013. URL: <http://dl.acm.org/citation.cfm?doid=2530268.2530272>, doi:
2285 10.1145/2530268.2530272.
- 2286 **219** Kento Sato, Naoya Maruyama, Kathryn Mohror, Adam Moody, Todd Gamblin, Bronis R
2287 de Supinski, and Satoshi Matsuoka. Design and modeling of a non-blocking checkpointing
2288 system. In *SC'12: Proceedings of the International Conference on High Performance Computing,*
2289 *Networking, Storage and Analysis*, pages 1–10. IEEE, 2012.
- 2290 **220** V. Savcenco, W. Hundsdorfer, and J.G. Verwer. A multirate time stepping strategy for stiff
2291 ordinary differential equations. *BIT*, 47:137–155, 2007.
- 2292 **221** Florian Schornbaum and Ulrich Rude. Massively parallel algorithms for the lattice Boltzmann
2293 method on nonuniform grids. *SIAM Journal on Scientific Computing*, 38(2):C96–C126, 2016.
- 2294 **222** Florian Schornbaum and Ulrich Rude. Extreme-scale block-structured adaptive mesh refine-
2295 ment. *SIAM Journal on Scientific Computing*, 40(3):C358–C387, 2018.
- 2296 **223** M. Schölzel. Reduced triple modular redundancy for built-in self-repair in vliw-processors. In
2297 *Signal Processing Algorithms, Architectures, Arrangements, and Applications SPA 2007*, pages
2298 21–26, Sep. 2007. doi:10.1109/SPA.2007.5903294.
- 2299 **224** Bruno Seny, Jonathan Lambrechts, Thomas Toulorge, Vincent Legat, and Jean-François
2300 Remacle. An efficient parallel implementation of explicit multirate Runge–Kutta schemes for
2301 discontinuous Galerkin computations. *Journal of Computational Physics*, 256:135–160, 2014.
- 2302 **225** F. Shahzad, J. Thies, M. Kreutzer, T. Zeiser, G. Hager, and G. Wellein. CRAFT: A library for
2303 easier application-level checkpoint/restart and automatic fault tolerance. *IEEE Transactions*
2304 *on Parallel and Distributed Systems*, 30(3):501–514, March 2019. doi:10.1109/TPDS.2018.
2305 2866794.
- 2306 **226** Faisal Shahzad, Jonas Thies, Moritz Kreutzer, Thomas Zeiser, Georg Hager, and Gerhard
2307 Wellein. CRAFT: A library for easier application-level checkpoint/restart and automatic
2308 fault tolerance. *IEEE Transactions on Parallel and Distributed Systems*, 30(3):501–514, 2018.
2309 doi:10.1109/TPDS.2018.2866794.
- 2310 **227** Manu Shantharam, Sowmyalatha Srinivasmurthy, and Padma Raghavan. Characterizing
2311 the impact of soft errors on iterative methods in scientific computing. *Proceedings of the*
2312 *international conference on Supercomputing - ICS '11*, page 152, 2011. URL: [http://portal.](http://portal.acm.org/citation.cfm?doid=1995896.1995922)
2313 [acm.org/citation.cfm?doid=1995896.1995922](http://portal.acm.org/citation.cfm?doid=1995896.1995922), doi:10.1145/1995896.1995922.
- 2314 **228** Richard V. Southwell. *Relaxation methods in engineering science, a treatise on approximate*
2315 *computation*. Oxford, Oxford Univ. Pr., 1946. 1.ed., reprint.
- 2316 **229** Robert Speck and Daniel Ruprecht. Toward fault-tolerant parallel-in-time integration with
2317 PFASST. *Parallel Computing*, 62:20–37, 2017.
- 2318 **230** P. Spitéri. Parallel asynchronous algorithms for solving boundary value problems. In M. Cosnard
2319 et al., editor, *Parallel Algorithms*, pages 73–84. North-Holland, 1986.
- 2320 **231** P. Spiteri. Parallel asynchronous algorithms: A survey. *Advances in Engineering Software*,
2321 149:1 – 34, 2020.
- 2322 **232** Linda Stals. *Parallel multigrid on unstructured grids using adaptive finite element methods*.
2323 PhD thesis, Department Of Mathematics, The Australian National University, Australia, 1995.
- 2324 **233** Linda Stals. Algorithm-based fault recovery of adaptively refined parallel multigrid grids. *The*
2325 *International Journal of High Performance Computing Applications*, 33(1):189–211, 2019.
- 2326 **234** J. Stearley, K. Ferreira, D. Robinson, J. Laros, K. Pedretti, D. Arnold, P. Bridges, and
2327 R. Riesen. Does partial replication pay off? In *IEEE/IFIP International Conference on*
2328 *Dependable Systems and Networks Workshops (DSN 2012)*, pages 1–6, June 2012. doi:
2329 10.1109/DSNW.2012.6264669.
- 2330 **235** Hana Straková, Gerhard Niederbrucker, and Wilfried N. Gansterer. Fault tolerance properties
2331 of gossip-based distributed orthogonal iteration methods. In Vassil N. Alexandrov, Michael Lees,

- 2332 Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M. A. Sloot, editors, *Proceedings*
 2333 *of the International Conference on Computational Science, ICCS 2013, Barcelona, Spain,*
 2334 *5-7 June, 2013*, volume 18 of *Procedia Computer Science*, pages 189–198. Elsevier, 2013.
 2335 doi:10.1016/j.procs.2013.05.182.
- 2336 **236** O. Subasi, J. Arias, O. Unsal, J. Labarta, and A. Cristal. Nan checkpoints: A task-based
 2337 asynchronous dataflow framework for efficient and scalable checkpoint/restart. In *2015 23rd*
 2338 *Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*,
 2339 pages 99–102, 2015.
- 2340 **237** O. Subasi, T. Martsinkevich, F. Zyulkyarov, O. Unsal, J. Labarta, and F. Cappello. Uni-
 2341 fied fault-tolerance framework for hybrid task-parallel message-passing applications. *The*
 2342 *International Journal of High Performance Computing Applications*, 32(5):641–657, 2018.
- 2343 **238** O. Subasi, G. Yalcin, F. Zyulkyarov, O. Unsal, and J. Labarta. A runtime heuristic to selectively
 2344 replicate tasks for application-specific reliability targets. In *2016 IEEE International Conference*
 2345 *on Cluster Computing (CLUSTER)*, pages 498–505, 2016.
- 2346 **239** O. Subasi, G. Yalcin, F. Zyulkyarov, O. Unsal, and J. Labarta. Designing and modelling
 2347 selective replication for fault-tolerant HPC applications. In *17th IEEE/ACM International*
 2348 *Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 452–457, May 2017.
 2349 doi:10.1109/CCGRID.2017.40.
- 2350 **240** Omer Subasi, Javier Arias, Osman Unsal, Jesus Labarta, and Adrian Cristal. Programmer-
 2351 directed partial redundancy for resilient HPC. In *Proceedings of the 12th ACM International*
 2352 *Conference on Computing Frontiers (CF)*, pages 47:1–47:2, New York, NY, USA, 2015. ACM.
 2353 URL: <http://doi.acm.org/10.1145/2742854.2742903>, doi:10.1145/2742854.2742903.
- 2354 **241** N. Sukhija, I. Banicescu, and F. M. Ciorba. Investigating the resilience of dynamic loop
 2355 scheduling in heterogeneous computing systems. In *Proceedings of the 14th International*
 2356 *Symposium on Parallel and Distributed Computing (ISPDC 2015)*, pages 194–203, June 2015.
 2357 doi:10.1109/ISPDC.2015.29.
- 2358 **242** Daniel B. Szyld. The mystery of asynchronous iterations convergence when the spectral radius
 2359 is one. Technical Report 98-102, Department of Mathematics, Temple University, Philadelphia,
 2360 Pa., October 1998. Available at <http://www.math.temple.edu/szyld>.
- 2361 **243** Daniel B. Szyld. Perspectives on asynchronous computations for fluid flow problems. In K. J.
 2362 Bathe, editor, *Computational Fluid and Solid Mechanics*, pages 377–380. Elsevier, 2001.
- 2363 **244** Daniel B. Szyld and Jian-Jun Xu. Convergence of some asynchronous nonlinear multisplitting
 2364 methods. *Numerical Algorithms*, 25:347–361, 2000.
- 2365 **245** X. Tang, J. Zhai, B. Yu, W. Chen, W. Zheng, and K. Li. An efficient in-memory checkpoint
 2366 method and its practice on fault-tolerant HPL. *IEEE Transactions on Parallel and Distributed*
 2367 *Systems*, 29(4):758–771, April 2018. doi:10.1109/TPDS.2017.2781257.
- 2368 **246** D. Tao, S. Di, Z. Chen, and F. Cappello. Significantly improving lossy compression for scientific
 2369 data sets based on multidimensional prediction and error-controlled quantization. In *2017*
 2370 *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1129–1139.
 2371 IEEE, 2017.
- 2372 **247** Dingwen Tao, Shuaiwen Leon Song, Sriram Krishnamoorthy, Panruo Wu, Xin Liang, Eddy Z.
 2373 Zhang, Darren J. Kerbyson, and Zizhong Chen. New-sum: A novel online ABFT scheme for
 2374 general iterative methods. In Hiroshi Nakashima, Kenjiro Taura, and Jack Lange, editors,
 2375 *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and*
 2376 *Distributed Computing, HPDC 2016, Kyoto, Japan, May 31 - June 04, 2016*, pages 43–55. ACM,
 2377 2016. iterative methods for linear systems: Krylov, stationary. doi:10.1145/2907294.2907306.
- 2378 **248** Containment Domain Team. Containment domains, 2014. URL: [https://lph.ece.utexas.](https://lph.ece.utexas.edu/public/CDs/ContainmentDomains)
 2379 [edu/public/CDs/ContainmentDomains](https://lph.ece.utexas.edu/public/CDs/ContainmentDomains).
- 2380 **249** Nanos team. Nanos++, 2020. URL: [https://www.bsc.es/research-and-development/](https://www.bsc.es/research-and-development/software-and-apps/software-list/nanos-rtl)
 2381 [software-and-apps/software-list/nanos-rtl](https://www.bsc.es/research-and-development/software-and-apps/software-list/nanos-rtl).
- 2382 **250** RAJA team. Raja performance portability layer. <https://github.com/LLNL/RAJA>, 2019.

- 2383 251 The Mercury Team. MERCURY programming language, 2020. URL: <https://www.mercurylang.org/>.
- 2384
- 2385 252 The Zoltan Team. Zoltan: Parallel Partitioning, Load Balancing and Data-Management
- 2386 Services, 2013. URL: <https://cs.sandia.gov/Zoltan/>.
- 2387 253 Keita Teranishi and Michael A. Heroux. Toward local failure local recovery resilience model
- 2388 using MPI-ULFM. In *Proceedings of the 21st European MPI Users' Group Meeting*, EuroMPI/ASIA'14, page 51–56, New York, NY, USA, 2014. Association for Computing Machinery.
- 2389 doi:10.1145/2642769.2642774.
- 2390
- 2391 254 Samuel Thibault. *On Runtime Systems for Task-based Programming on Heterogeneous Plat-*
- 2392 *forms*. Habilitation à diriger des recherches, Université de Bordeaux, December 2018. URL:
- 2393 <https://hal.inria.fr/tel-01959127>.
- 2394 255 G. Tumolo and L. Bonaventura. A semi-implicit, semi-Lagrangian discontinuous Galerkin
- 2395 framework for adaptive numerical weather prediction. *Quarterly Journal of the Royal Meteorological Society*, 141(692):2582–2601, 2015.
- 2396
- 2397 256 Henk A. van der Vorst and Qiang Ye. Residual replacement strategies for Krylov subspace
- 2398 iterative methods for the convergence of true residuals. *SIAM Journal on Scientific Computing*,
- 2399 22(3):835–852, 2000.
- 2400 257 Jyothish Varma, Chao Wang, Frank Mueller, Christian Engelmann, and Stephen L. Scott.
- 2401 Scalable, fault-tolerant membership for MPI tasks on HPC systems. In *Proceedings of the*
- 2402 *20th ACM International Conference on Supercomputing (ICS) 2006*, pages 219–228, Cairns,
- 2403 Australia, June 28–30, 2006. ACM Press, New York, NY, USA. doi:10.1145/1183401.
- 2404 1183433.
- 2405 258 J. Vignes. Discrete Stochastic Arithmetic for validating results of numerical software. *Numerical*
- 2406 *Algorithms*, 37(1–4):377–390, December 2004.
- 2407 259 T. N. Vijaykumar, Irith Pomeranz, and Karl Cheng. Transient-fault recovery using simultaneous
- 2408 multithreading. In *In proceedings of the 29th annual international symposium on computer*
- 2409 *architecture*, pages 87–98. IEEE Computer Society, 2002.
- 2410 260 John von Neumann. First Draft of a Report on the EDVAC, 1945. URL:
- 2411 [https://nsu.ru/xmlui/bitstream/handle/nsu/9018/2003-08-TheFirstDraft.pdf?](https://nsu.ru/xmlui/bitstream/handle/nsu/9018/2003-08-TheFirstDraft.pdf?sequence=1&isAllowed=y)
- 2412 [sequence=1&isAllowed=y](https://nsu.ru/xmlui/bitstream/handle/nsu/9018/2003-08-TheFirstDraft.pdf?sequence=1&isAllowed=y).
- 2413 261 John von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable
- 2414 components. *Automata Studies*, pages 43–98, 1956.
- 2415 262 Chao Wang, Frank Mueller, Christian Engelmann, and Stephen L. Scott. A job pause
- 2416 service under LAM/MPI+BLCR for transparent fault tolerance. In *Proceedings of the 21st*
- 2417 *IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2007*, pages
- 2418 1–10, Long Beach, CA, USA, March 26–30, 2007. ACM Press, New York, NY, USA. doi:
- 2419 10.1109/IPDPS.2007.370307.
- 2420 263 Chao Wang, Frank Mueller, Christian Engelmann, and Stephen L. Scott. Hybrid check-
- 2421 pointing for MPI jobs in HPC environments. In *Proceedings of the 16th IEEE Interna-*
- 2422 *tional Conference on Parallel and Distributed Systems (ICPADS) 2010*, pages 524–533,
- 2423 Shanghai, China, December 8–10, 2010. IEEE Computer Society, Los Alamitos, CA, USA.
- 2424 doi:10.1109/ICPADS.2010.48.
- 2425 264 Chao Wang, Frank Mueller, Christian Engelmann, and Stephen L. Scott. Proactive process-level
- 2426 live migration and back migration in HPC environments. *Journal of Parallel and Distributed*
- 2427 *Computing (JPDC)*, 72(2):254–267, February 2012. doi:10.1016/j.jpdc.2011.10.009.
- 2428 265 Chris Weaver and Todd M. Austin. A fault tolerant approach to microprocessor design.
- 2429 In *Proceedings of the 2001 International Conference on Dependable Systems and Networks*
- 2430 *(Formerly: FTCS)*, USA, 2001. IEEE Computer Society.
- 2431 266 Jordi Wolfson-Pou and Edmond Chow. Asynchronous multigrid methods. *2019 IEEE*
- 2432 *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 101–110, 2019.
- 2433 267 Panruo Wu and Zizhong Chen. FT-ScaLAPACK: Correcting soft errors on-line for ScaLAPACK
- 2434 Cholesky, QR, and LU factorization routines. In *Proceedings of the 23rd International*

- 2435 *Symposium on High-performance Parallel and Distributed Computing*, HPDC '14, pages 49–60,
 2436 New York, NY, USA, 2014. ACM. URL: <http://doi.acm.org/10.1145/2600212.2600232>,
 2437 doi:10.1145/2600212.2600232.
- 2438 **268** Panruo Wu, Qiang Guan, Nathan DeBardeleben, Sean Blanchard, Dingwen Tao, Xin Liang,
 2439 Jieyang Chen, and Zizhong Chen. Towards practical algorithm based fault tolerance in dense
 2440 linear algebra. In *Proceedings of the 25th ACM International Symposium on High-Performance*
 2441 *Parallel and Distributed Computing*, New York, NY, USA, 2016. Association for Computing
 2442 Machinery. dense linear algebra. doi:10.1145/2907294.2907315.
- 2443 **269** Y. Kim, J. S. Plank, and J. J. Dongarra. Fault tolerant matrix operations using checksum and
 2444 reverse computation. In *Proceedings of 6th Symposium on the Frontiers of Massively Parallel*
 2445 *Computation (Frontiers '96)*, pages 70–77, Oct 1996. doi:10.1109/FMPC.1996.558063.
- 2446 **270** Ichitaro Yamazaki, Edmond Chow, Aurelien Bouteiller, and Jack J. Dongarra. Performance of
 2447 asynchronous optimized Schwarz with one-sided communication. *Parallel Comput.*, 86:66–81,
 2448 2019. doi:10.1016/j.parco.2019.05.004.
- 2449 **271** Yonghong Yan, Jisheng Zhao, Yi Guo, and Vivek Sarkar. Hierarchical place trees: A portable
 2450 abstraction for task parallelism and data movement. In *International Workshop on Languages*
 2451 *and Compilers for Parallel Computing*, pages 172–187. Springer, 2009.
- 2452 **272** Ulrike Yang, Piotr Luszczek, Satish Balei, and Keita Teranishi. An introduction to the xsdk a
 2453 community of diverse numerical hpc software packages. Technical report, Sandia National
 2454 Lab.(SNL-CA), Livermore, CA (United States), 2019. doi:10.6084/m9.figshare.7779452.
 2455 v1.
- 2456 **273** J. Yu, D. Jian, Z. Wu, and H. Liu. Thread-level redundancy fault tolerant CMP based on
 2457 relaxed input replication. In *2011 6th International Conference on Computer Sciences and*
 2458 *Convergence Information Technology (ICCIT)*, pages 544–549, Nov 2011.
- 2459 **274** Gengbin Zheng, Xiang Ni, and Laxmikant V Kalé. A scalable double in-memory checkpoint
 2460 and restart scheme towards exascale. In *IEEE/IFIP International Conference on Dependable*
 2461 *Systems and Networks Workshops (DSN 2012)*, pages 1–6. IEEE, 2012.
- 2462 **275** Ziming Zheng, Andrew A. Chien, and Keita Teranishi. Fault tolerance in an inner-outer
 2463 solver: A GVR-enabled case study. In Michel J. Daydé, Osni Marques, and Kengo Nakajima,
 2464 editors, *High Performance Computing for Computational Science - VECPAR 2014 - 11th*
 2465 *International Conference, Eugene, OR, USA, June 30 - July 3, 2014, Revised Selected Papers*,
 2466 volume 8969 of *Lecture Notes in Computer Science*, pages 124–132. Springer, 2014. doi:
 2467 10.1007/978-3-319-17353-5_11.
- 2468 **276** G Zoutendijk. Nonlinear programming, computational methods. *Integer and nonlinear*
 2469 *programming*, pages 37–86, 1970.