This is an accpeted manuscript accepted in International Journal of High Performance Computing Applications. The published version can be founf at: https://doi.org/10.1177/10943420211055188

Resiliency in Numerical Algorithm Design for Extreme Scale Simulations

Journal Title
XX(X):3–66

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Emmanuel Agullo[1], Mirco Altenbernd[2], Hartwig Anzt[3], Leonardo Bautista-Gomez[4], Tommaso Benacchio[5], Luca Bonaventura[5], Hans-Joachim Bungartz[6], Sanjay Chatterjee[7], Florina M. Ciorba[8], Nathan DeBardeleben[9], Daniel Drzisga[6], Sebastian Eibl[10], Christian Engelmann[11], Wilfried N. Gansterer[12], Luc Giraud[1], Dominik Göddeke[2], Marco Heisig[10], Fabienne Jézéquel[13], Nils Kohl[10], Xiaoye Sherry Li[14], Romain Lion[15], Miriam Mehl[2], Paul Mycek[16], Michael Obersteiner[6], Enrique S. Quintana-Ortí[17], Francesco Rizzi[18], Ulrich Rüde[10,16], Martin Schulz[6], Fred Fung[19], Robert Speck[20], Linda Stals[19], Keita Teranishi[21], Samuel Thibault[15], Dominik Thönnes[10], Andreas Wagner[6] and Barbara Wohlmuth[6]

Prepared using sagej.cls

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.

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

Acronyms

ABFT *Algorithm-Based Fault Tolerance*. 22, 36, 37

AMR *Adaptive Mesh Refinement*. 25, 26, 33

API *Application Programming Interface*. 11, 15, 16

BLCR *Berkeley Lab Checkpoint/Restart*. 9

CDs *Containment Domains*. 14

CPU *Central Processing Unit*. 11

CRC *Cyclic Redundancy Checks*. 8

DLS *Dynamic Loop Self-scheduling*. 16

DLS4LB *Dynamic Loop Scheduling for Load Balancing*. 14

[1]Inria
[2]Universität Stuttgart
[3]KIT – Karlsruher Institut für Technologie
[4]Barcelona Supercomputing Center
[5]Politecnico di Milano
[6]TU München
[7]NVIDIA Corporation
[8]Universität Basel
[9]Los Alamos National Laboratory
[10]Universität Erlangen-Nürnberg
[11]Oak Ridge National Laboratory
[12]University of Vienna
[13]Université Paris 2 – Paris
[14] Lawrence Berkeley National Laboratory
[15]University of Bordeaux
[16]Cerfacs
[17]Universitat Politècnica de València
[18]NexGen Analytics
[19]Australian National University
[20]Jülich Supercomputing Centre
[21]Sandia National Laboratories – California

Corresponding author:
Linda Stals, Mathematical Sciences Institute, Australian National University, Canberra ACT, 2601, Australia
Email: linda.stals@anu.edu.au

Prepared using sagej.cls *[Version: 2017/01/17 v1.20]*

DMR *Double modular redundancy*. 21

DRAM *Dynamic Random-Access Memory*. 13

DUE *Detectable, but Uncorrectable Error*. 32

ECC *Error Correcting Codes*. 8, 11, 32

FEM *Finite Element Method*. 34, 36, 37

FFT *Fast Fourier Transforms*. 18

FPGA *Field-Programmable Gate Arrays*. 15

GPU *Graphics Processing Units*. 11, 15, 33

GVR *Global View Resilience*. 14

HBM *High-Bandwidth Memory*. 13

HPC *High Performance Computing*. 5, 6, 9, 11, 12, 13, 15, 16, 32, 33

LFLR *Local-Failure Local-Recovery*. 21, 22, 26, 27

MDS *Meta Data Service*. 11

MPI *Message Passing Interface*. 9

NVM *Non-Volatile Memory*. 13

ODE *ordinary differential equations*. 19, 20, 36

OS *Operating Systems*. 8, 33

PCG *Preconditioned Conjugate Gradient*. 19, 23

PDE *Partial Differential Equations*. 19, 20, 21, 24, 36

PFS *Parallel File System*. 7, 11, 13

PMPI *MPI Profiling Interface*. 11

PVFS *Parallel Virtual File System*. 11

QOS *Quality of Service*. 11

rDLB *robust Dynamic Load Balancing*. 16

SDC *Silent Data Corruption*. 8, 9, 12

SSD *Solid-State Drives*. 13

TMR *Triple Modular Redundancy*. 21

ULFM *User Level Failure Mitigation*. 13, 15

1 Introduction

Numerical simulation is the third pillar in science discovery at the same level as theory and experiments. To cope with the ever demanding computational resources needed by complex simulations, the computational power of high performance computing systems continues to increase by using an ever larger number of cores or by specialized processing. On the technological side, the continuous shrinking of transistor geometry and the increasing complexity of these devices affect their sensitivity to external effects and thus diminish their reliability. A direct consequence is that *High Performance Computing* (HPC) applications are increasingly prone to errors. Therefore the design of resilient systems and numerical algorithms that are able to exploit possible unstable HPC platforms has became a major concern in the computational science community. To tackle this critical challenge on the path to extreme scale computation an holistic and multidisciplinary approach is required that needs to involve researchers from various scientific communities ranging from the hardware/system community to applied mathematics for the design of novel numerical algorithms. In this article, we summarize and report on the outcomes of a Dagstuhl seminar held March 1-6, 2020,* on the topic *Resiliency in Numerical Algorithm Design for Extreme Scale Simulations*. We should point out that, although error and resiliency was already quoted by J. von Neumann in his first draft report on EDVAC [2, P.1, Item 1.4], it became again a central concern for the HPC community in the late 2000' when the availability of the first exascale computers was envisioned for the forthcoming decades. In particular, several workshops were organized in the IESP (International Exascale Software Project) and EESI (European Exascale Software Initiative) framework [3].

The hardware/system resilience community has previously defined terminology related to how faults, errors, and failures occur on computing systems [4]. In this article our focus is less on the cause of an error (or the underlying fault), and more on how an error presents itself at the algorithmic level (or layer), impacting algorithms and applications. We thus simplify the terminology often used in the hardware resilience and fault-tolerance community by not using terms like soft error or hard error, and generally do not concern ourselves with the reproducibility of an error (e.g., transient, intermittent or permanent). This abstraction keeps the algorithmic techniques discussed herein general and applicable to a variety of fault models, current architectures, and hopefully of use in future technologies.

*https://www.dagstuhl.de/en/program/calendar/semhp/?semnr=20101

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

Many computing-intensive scientific applications that are dependent on HPC performance 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 [5, 6]. 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

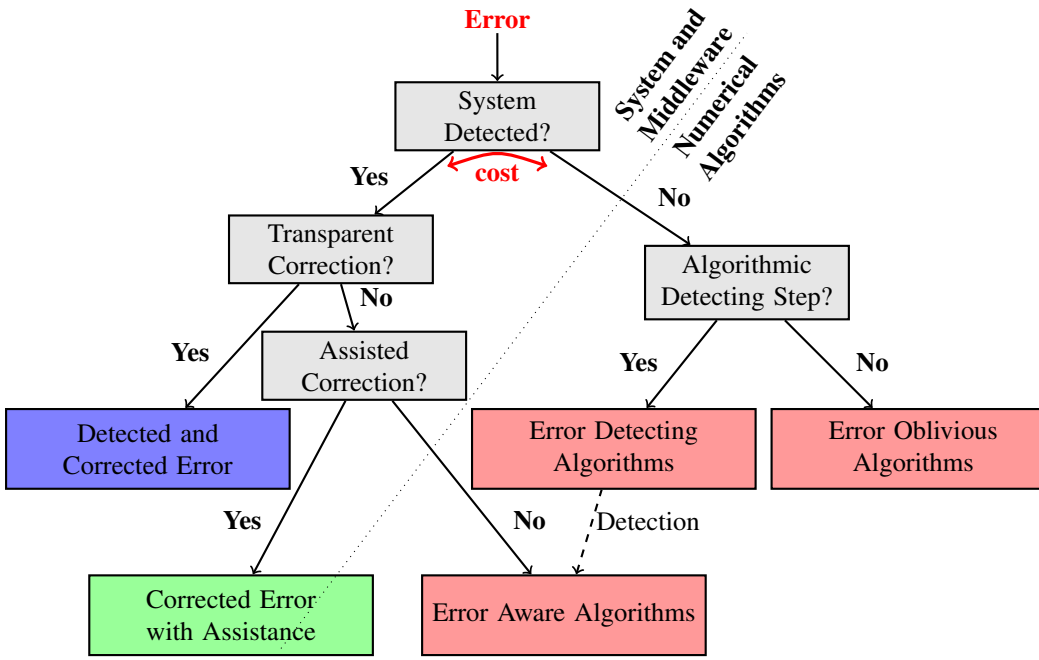

Figure 1. A classification of error handling.

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 an 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. [7].

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 [8]. 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 3s to 6s 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) [9] is a system-level checkpoint/restart solution that transparently saves and restores process state. In conjunction with a *Message Passing Interface* (MPI) [10] implementation, it can transparently save and restore the process states of an entire MPI application. An extension of BLCR [11–13] 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 [14] 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.

Silent Data Corruption (SDC) detection and protection FlipSphere [15] is a tunable, transparent *Silent Data Corruption* (SDC) detection and correction library for HPC applications. It offers comprehensive SDC protection for application program memory using on-demand memory page integrity verification. Experimental benchmarks show that it can protect 50% to 80% of program memory with time overheads of 7% to 55%. Other data-prediction based SDC detection methods have been proposed such as Adaptive Impact Driven SDC detector [16], MACORD [17] or for end-to-end detection for protecting lossy compression in [18]. Note that the data-prediction based SDC detectors can detect not only memory errors but also computation errors to a certain extent in principle. They can also protect the errors by combining the checkpoint-restart tools such as FTI.

Proactive fault tolerance using process or virtual machine migration Proactive fault tolerance [19–21] prevents compute node failures from impacting running applications by migrating parts of an application, i.e., tasks, processes, or virtual machines, away from nodes that are about to fail. Pre-fault indicators, such as a significant increase in temperature, can be used to avoid an imminent failure through anticipation and reconfiguration. As computation is migrated away, application failures are avoided,

which is significantly more efficient than checkpoint/restart if the prediction is accurate enough. The proactive fault tolerance framework consists of process and virtual machine migration, scalable system monitoring and online/offline system health analysis. The process-level live migration supports continued execution of applications during much of process migration and is integrated into an MPI execution environment. Experiments indicate that 1s to 6.4s of prior warning are required to successfully trigger live process migration, while similar operating system virtualization mechanisms require 13s to 24s. This error oblivious approach complements checkpoint/restart by nearly cutting the number of checkpoints by half when 70% of the faults are handled proactively.

Resiliency using task-based runtime systems Task-based runtime systems have appealing intrinsic features for resiliency due to the fault isolation they provide by design as they have a view of the task flow and dynamically schedule task on computing units (often to minimize the time to solution or energy consumption). Once an error is detected and identified by the hardware or the algorithm, the runtime system can limit its propagation through the application by reasoning about the data dependencies among tasks [22]. For example, one can envision the scenario where an uncorrectable hardware error is detected triggering the runtime system to dynamically redistribute the tasks to the remaining resources available.

Task-based runtime systems can also limit the size of the state needed to be saved to enable restarting computations, when an error is encountered [23–25]. In classical checkpoint-restart mechanisms, the size of the checkpoint can become very large for large-scale applications, and managing it can take up a significant portion of the overall execution. In the task-based programming model, each checkpoint is a cut in the task graph, which can be expressed trivially within the task submission code, and only the data of the crossing edges need to be saved. Even better, the synchronization between the management of checkpoint data and application execution can be greatly relaxed. The transfer of the data to the checkpoint storage can indeed be started as soon as the data is produced within the task graph. It is not necessary to wait for all of the tasks before the checkpoint to be completed. A checkpoint is finalised when all its pieces of data have been collected. It is possible that tasks occurring after the checkpoint may run to completion before the checkpoint itself is finished. Further, identification of idempotent tasks can greatly help task-based runtimes to further reduce the overheads by completely avoiding data backups specific to those tasks.

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.

Recent works on on-node task parallel programming models suggest that a simple extension of the existing task-based programming framework enables efficient localized recovery [26–28].

Parallel programming models such as Charm++ [29], HClib [30], HPX [31], OmpSs [32] and StarPU [23] integrate a variety of resilient task program execution

options such as replay, replication, algorithm-based fault tolerance and task-based checkpointing.

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 [33] offers checkpoint/restart for computation offloaded to GPU using OpenCL [34]. 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 [35] using a novel concept for *Quality of Service* (QOS) and a corresponding *Application Programming Interface* (API) [36]. 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 [37–40]. 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 [41] often offer some type of active/standby fail-over mechanism for the MDS. A solution [42] 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 [43–46]. Adaptive partial redundancy has also been proposed wherein a subset of processes is dynamically selected for replication [47]. 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 [48–50]. 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 [51].

Resilience via complete and/or partial redundancy RedMPI [52] 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 [43]. 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 [53–56]. 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 [57–60]. 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 [61,62]. 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 [63], 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 [64], a resilient key-value blob-storage. Other resilient checkpointing techniques include the self-checkpoint technique [65], which reduces common redundancies while writing checkpoints, or techniques reducing the amount of required memory through hierarchical checkpointing [66], or differential checkpointing [67].

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. [68]. 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. [69] 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 [70]. 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 [71].

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 [72], FTI [59], CRAFT [73] and VeloC [74]. 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 SCR. 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 [75]. 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) [76]. 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) [77] 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 [78].

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 [79] 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) [80], a recently developed library for MPI applications that contains a portfolio of self-scheduling based algorithms for load balancing. StarPU [23] proposes support for asynchronous load-balancing [24] 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 [32] 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 [81] and the nanos++ runtime system [82].

HCLib [30] 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++ [29] 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 [83] or MPI-ULFM [84] to enable alternatives of global shut-down, as well as better error handling extensions, like MPI_Reinit [85], 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 [68, 86–89]. 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 [90]. 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 [91]. 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 achieve resilience. rDLB is integrated into a dynamic loop scheduling library (DLS4LB, see Section 2.2.1) for MPI applications. rDLB enables the tolerance of up to $(P-1)$ process failures, where P is the number of processes executing an application. For execution environments with performance-related fluctuations, rDLB boosts the robustness of *Dynamic Loop Self-scheduling* (DLS) techniques by a factor up to 30 and decreases application execution time up to 7 times compared to their counterparts without rDLB.

Resilience extension for performance portable programming abstractions With the increasing diversity of the node architecture of HPC systems, performance portability has become an important property to support a variety of computing platforms with the same source code while achieving a comparative performance to those programmed with the platform specific programming models. Today, Kokkos [92] and Raja [93, 94] accommodate modern C++ APIs to permit an abstraction of data allocation and parallel loop execution for a variety of runtime software and node architectures. This idea can be extended to express the redundancy of data and computation to achieve resilience while hiding the details of the data persistence and redundant computation. Kokkos' resilience extension provides (1) redundant execution for parallel loops and (2) automatic checkpointing of Kokkos' data objects. For checkpointing, all Kokkos data objects are registered to the underlying checkpoint backend (C++ I/O, MPI I/O and VeloC [74]) and the status of individual view instances are monitored by the Kokkos runtime. For invoking checkpointing, Kokkos provides C++ Lambda to delineate the program source to indicate (1) when checkpoint is performed and (2) where the checkpointed data is recovered when re-executed. More information on the resilient Kokkos can be found in [95]

Software engineering approaches for resilience by design Resilience design patterns [96, 97] 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 [98] offers multi-resilience for detection, containment and mitigation of silent data corruption and MPI process failures.

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	×		×	×	×	×

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

BLAS 2/3: Checksum encoding matrices, introduced by Huang and Abraham [99] requires (i) adding redundant data in some form (encoding), (ii) redesign of the algorithm to operate on the respective data structures (processing), and (iii) checking the encoded data for errors (detection). We ignore the recovery phase here and refer to Section 3.2. Checksums are used in FT-ScaLAPACK [100] for dense matrix operations such as MM, LU and QR factorization and more recently in hierarchical matrix multiplication [76]. Wu et al. give a good survey of checksum deployment in dense linear algebra [101].

Gauss-Seidel/Jacobi/SOR and multigrid: In [102], checksums are used to detect errors in the Jacobi smoother, the restriction and interpolation operators of a multigrid method solving a two-dimensional Poisson equation.

Krylov subspace methods: Tao et al. propose a new checksum scheme using multiple checksum vectors for sparse matrix-vector multiplication, which is shown to be generally effective for several preconditioned Krylov iterative algorithms [103]. Also [104,105] use checksums for protection within the conjugate gradient (CG) algorithm.

FFT: Checksum can also be used in *Fast Fourier Transforms* (FFT)s similarly as in matrix-vector multiplication. Liang et al. [106] develop a new hierarchical checksum scheme by exploiting the special discrete Fourier transform matrix structure and employ special checksum vectors. Checksums are applicable to many important kernels such as matrix-matrix multiplication, but are costly. In addition, it can be difficult to specify a suitable threshold for 'equality' in the presence of round-off errors. For many numerical calculations such as scalar products, checksums are not applicable at all.

CNN: Convolutional Neural Networks are emerging as a promising novel component for large scale simulations where hybridizing classical numerical algorithms and machine learning techniques can be considered as an alternative to reduce some computational effort and to better exploit the massive data produced by large simulations. Extending ABFT principles to CNN has recently being investigated in [107] to detect and correct CNN inference process.

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 [108], Hestenes and Stiefel noticed that the reciprocal value of α (the step length) is bounded above (repectively, 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 [105].

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 [109]. 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 [110] and have been considered in the context of Krylov subspace methods in finite precision arithmetic [111, 112].

Vorst and Ye proposed a residual gap bound [113] (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 [105] when bit flips occur. The detection of errors in Krylov methods via violation of orthogonality is proposed in [114].

Multigrid: Calhoun et. al [115] 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

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

when solving the linear system $A\boldsymbol{x} = \boldsymbol{b}$ 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 [116].

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 [117]. In the context of parallel-in-time integration with parareal, consecutive iterates are considered in [118] to detect errors in the solution vector. In [119], 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 [120] 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 [121] 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 [122], quadrature [123–125], solvers for PDEs [126, 127], or machine learning tasks [128–130] (e.g., classification, regression, clustering, or density estimation). One particular incarnation of sparse grid methods is the so-called combination technique [131]. 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 [132–137]. 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 [138, 139] or TMR [140, 141]. In the second approach the computations are repeated twice or thrice on the same resource [142, 143]. 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 [144] 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 [145] use replication of fault-prone components as an error detection technique in a multigrid method. Also error detection in the time stepping methods from [119] 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 [146]. 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 [147, 148] 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) [88]. This model provides application developers with the ability to recover locally and continue application execution when a process is lost. In [88], 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.

Original algorithm-based fault tolerance with checksums The term *Algorithm-Based Fault Tolerance* (ABFT) was originally coined in conjunction with protecting matrix operations with checksums to handle bit flips [149], mostly assuming exact arithmetic calculation for detection and mitigation. (See Section 3.1.2 for a more detailed discussion on checksums). The main drawback of checksums is that only limited error patterns can be corrected and its robust practical implementation in finite precision arithmetic can be complicated to tune to account for round-off errors. A second drawback is that the checksum encoding, detection and recovery methods are specific to a particular calculation. A new scheme needs to be designed and proved mathematically for each new operation. A further drawback is to tolerate more errors, more encoded data is needed, which may be costly both in memory and in computing time.

ABFT concepts have been extended to process failures for a wide range of matrix operations both for detection and mitigation purposes [150–154] and general communication patterns [155]. ABFT has also recently been proposed for parallel stencil-based operations to accurately detect and correct silent data corruptions [156]. In these scenarios the general strategy is a combination of checkpointing and replication of checksums. In-memory checkpointing [154] can be used to improve the performance. The main advantage of these methods is their low overhead and high scalability.

In practice, the significance of a bit flip strongly depends on its location, i.e., which bit in the floating point representation is affected. Classical ABFT has been extended to take into account floating point effects in the fault detection (checksums in finite precision) as well as in the fault correction and to recover from undetected errors (bit flips) in all positions without additional overhead [157].

Iterative linear solvers Iterative linear solvers based on fixed point iteration schemes are, in general, examples of error oblivious algorithms, as described in Section 3.3. The convergence history of the scaled residual norm observed within the iterative scheme often resembles the curves displayed in Figure 2. In this case the iterative scheme is a multigrid method, as in [158, 159]. The peaks in the residual occur after data has been lost and when the iterations are allowed to restart with some form of replacement of the lost data. In the simplest case, the lost data may just be re-initialized with the value of zero, and recovery techniques to obtain better solutions are discussed in Section 3.2.2.

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

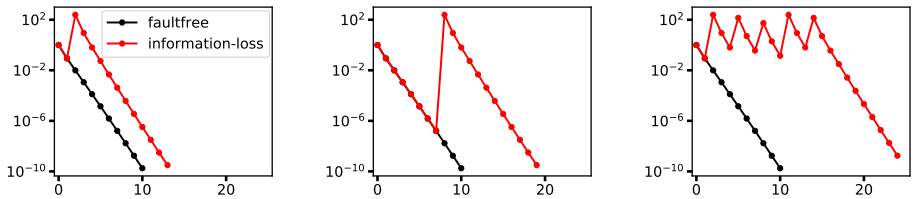

Figure 2. Convergence history of the residual norm as a function of the iteration count for three examples of information loss. From left to right: early, late, and multiple times

Explicit recovery at the algorithmic level from undetected errors have been studied for iterative linear solvers [160]. In contrast to restarting, a number of algorithm based recovery strategies have been proposed, including approximate or heuristic interpolation methods [161]. An approach of exactly recovering the state of the iterative solver before the node failure has been investigated for the *Preconditioned Conjugate Gradient* (PCG) and related methods [162, 163]. This also includes studying scenarios with multiple simultaneous node failures [164] and scenarios where no replacement nodes are available [165].

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 [161, 166, 167] for some illustrations on those numerical remedies in a parallel distributed memory framework and to [168] 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 [169]. They can also be extended to the time domain in the context of multilevel parallel-in-time integration techniques [170].

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

Adaptive recovery techniques for extreme scale multigrid Some of the most efficient solvers of PDE, such as parallel geometric multigrid methods [171, 172], can be based on the exchange of ghost layers in a non-overlapping domain partitioning. This automatically leads to a redundancy in interface data between subdomains that in turn permits the design of an efficient two-step recovery strategy for iterative solvers. This is of particular interest in large-scale parallel computations. When each subdomain is large, then the ratio between the data on its surface and the volume data in its interior becomes small.

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

After re-initialization, the corrupted and reinitialized data could pollute the solution globally, meaning that the locally increased error in the faulty domain can spread globally and thus also affect the healthy subdomain. In order to avoid this pollution, the communication between the healthy and faulty sub-problems is interrupted during the second step of the recovery process. In the second step, we continue with the original

iterative solver restricted to the healthy sub-problem and select a suitable one for the faulty one. After some number of asynchronous iteration steps both sub-problems are reconnected, see [159], and the global iterative solver strategy is resumed. Note that the reconnecting step is mandatory for the convergence of the iterative solver. The tearing step separating the subdomains is mandatory to preserve the accuracy of the dynamic data in the healthy sub-problem, and without this step the corrupted data from the faulty sub-domain pollutes the global solution. Of critical importance for the performance of the method are the accuracy of the faulty sub-problem solver at re-connection time and the time spent in the recovery mode. In the faulty domain, the lost data can be initialized with 0, or, alternatively, compressed checkpointed data can be used as described in the following section on compression techniques for checkpoint-restart. Note, however, that with straight-forward compression techniques, compressed checkpoint data will only be useful to recover the low frequency components in the faulty domain. If the local recovery is performed with multigrid, then the low frequencies are in any case cheap to recover, as compared to the cost of recomputing the lost high frequency components.

The accuracy within a multigrid strategy can be easily controlled by a hierarchical sum of weighted residuals [173]. The overhead cost for the a-posterior error indicator is quite small compared to the overall solver cost. Having an estimate for the algebraic error in both sub-problems at hand, the re-connection step is determined automatically. To speed up the time which is spent in the recovery, a so-called 'superman strategy' is applied [159], 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 [174], 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 [175]. 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 [159, 174]. 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 [176–178] 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

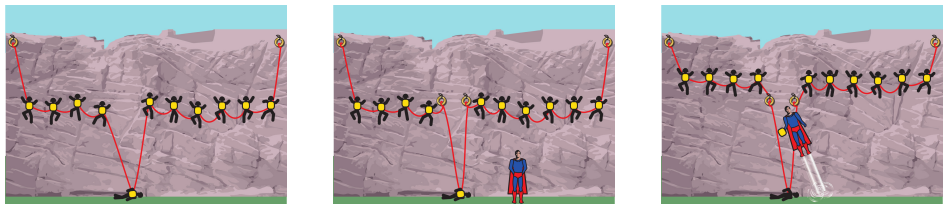

Figure 3. Illustration of the steps in the adaptive recovery technique for extreme scale multigrid. Left: A detectable error occurred. Middle: The communication between the healthy and faulty sub-domains is interrupted. Right: The original iterative solver restricted to the healthy domain continues while 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.

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 [57]. 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 Kryloy-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 [179], the SZ library [180–182] 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. We mention that lossy compression to alleviate the cost of checkpointing in iterative methods is also considered in [183].

Note that multigrid can be interpreted as a lossy compression technique in itself, with a number of mathematical peculiarities that need consideration [158]. 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 [179]. 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 [179].

Resilience with sparse grids Resilience can be added on various abstraction levels of the algorithm. For PDE problems one traditionally adds resilience on the level of linear algebra operations, on the solver level for linear/non-linear equations, or on the time-stepping algorithm. However, this may in some cases not be coarse-grained enough to minimize the overhead of resilience techniques, especially when errors occur rarely. In [126,132,135–137,184] the authors demonstrate a fault-tolerant framework for solving high-dimensional PDEs that applies fault tolerance on top of the individual PDE solver. The framework boosts the scalability of black-box PDE solvers while making it simultaneously resilient to faults by applying the sparse grid combination technique. In this technique the PDE simulation is distributed over many coarse grids, which can be processed in parallel. At regular intervals the results of these grids are combined to

obtain the final sparse grid result. In presence of faults the affected grids can be neglected and an alternative combination scheme is calculated via an optimization routine. If too many grids are lost, the last combination result serves as an in-memory checkpoint to recompute the required grids. In [132] it is shown that this lossy recovery provides very good results even with high error frequencies. At the same time the parallel efficiency is only slightly affected.

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

We refer to [185] for a parallel adaptive multigrid method that uses a sophisticated dynamic data structures to store a nested sequence of meshes and the iterative evolving solution. Stals demonstrates that it is possible to implement a fault recovery procedure that builds on the original parallel adaptive multigrid refinement algorithm [186] in the case of a fail-stop fault. It is assumed that a copy of the coarsest grid can always be accessed after a fault has occurred, i.e., it is stored off the processor. The challenge in recovering an adaptively refined grid is that the mesh distribution changes during any load balancing procedures, i.e., the local information that was available during the original refinement process will have been modified or removed. Nevertheless it is demonstrated that the neighbouring healthy processors contain enough intact information so that the necessary structure can be recovered to pass into the refinement routine. In the case of uniform refinement, the original multilevel grid is recovered. In the case of an adaptively refined grid, enough of the structure is recovered to re-establish the correct communication pattern allowing the solution process to run to completion, but potentially with reduced accuracy. The neighbouring healthy processors will only contain enough information to guide the refinement around the edge of the recovered subgrid. Further refinement within the interior of the recovered subgrid may be required to improve the accuracy of the solution.

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

3.3 Error oblivious algorithms

In this section, we give examples of algorithms that are error oblivious in the sense that they can recover without assistance from errors that do not occur too frequently. For example, many fixed point iterative solvers are able to execute to completion if, e.g., a bit flip error occurs in the solution vector. However, every error likely increases the execution

time of the algorithm. We thus define two quality criteria for error oblivious algorithms and use to assess the examples in the remainder of this section: (i) correctness, and (ii) efficiency in terms of execution time.

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

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

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

3.3.1 Gossip based methods A potentially interesting alternative in large-scale parallel environments that does not require any explicit error detection mechanisms utilizes gossip-based methods and their inherent resilience properties. Such algorithms by nature build up redundancy in the system and can thus can efficiently recover automatically from various types of faults/errors without any need to explicitly detect them. In particular, Gansterer et al. have studied and extended the resilience of gossip-based aggregation and reduction methods [187–189]. Based on these building blocks, they have developed and analyzed several more complex resilient numerical algorithms, such as orthogonalization methods [189, 190], eigensolvers [191], and least squares solvers [192].

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

3.3.2 Fixed-point methods We view fixed-point methods as methods that converge globally when certain conditions are satisfied. For example, the Jacobi iterative schemes will converge for any initial guess if the matrix is diagonally dominant. Fixed-point based iterative methods are by design resilient to bit flips. However, the convergence delay can be significant. Anzt et al. [193, 194] propose techniques improving the cost-robustness with little overhead.

A class of numerical algorithms that by design have properties attractive for resilience are asynchronous iterative methods [195–202]. In order to avoid misunderstandings, we point out that this class of methods is unrelated to the idea of asynchronous dynamic load balancing [203] as addressed in Section 2.1. Instead, asynchronous iterative methods, stemming from the concept of chaotic iterations [204], 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 [205]. Chow et al. [206, 207] developed an asynchronous algorithm for generating incomplete factorizations, Coleman et al. [208] 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 [209–213].

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 [215] and is controlled by efficient hierarchical error estimators [173]. The parallel implementation [216] will automatically correct errors. More recently, asynchronous methods have been proposed for nonlinear multi-splitting [217] and eigenvalue computations like Google's Pagerank algorithm [218]. More recently, also the idea of asynchronously solving coarse-grid error correction equations has been investigated, leading to an asynchronous multigrid algorithm [219]. 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 The authors of [5] 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 [220] the authors use a slightly different terminology and call their method numerically self-stabilizing, a term which originates in the context of distributed systems [221]. They introduce two error oblivious [221] 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 [222] 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.

A recent study [179] showed that storing both the most recent iterate and search direction can improve the CG recovery. Furthermore, CG recovery is viable for systems with low fault rates, but for high fault rates GMRES is more robust to errors and local recovery.

The efficiency of a Krylov subspace solver is heavily reliant on the preconditioner. That being the case, it is prudent to ask how the solver will be impacted if the preconditioner is compromised. Elliott [223, 224] investigates a nested solver strategy that combines an unreliable, but cheap, inner solver with a reliable expensive outer solver to minimize the numerical impact of soft errors. The CG and GMRES solvers are evaluated with the algebraic multigrid as the preconditioner. Coleman et al. [208] propose an alternative approached by developing a fault tolerant incomplete ILU algorithm.

3.3.4 Domain decomposition In [225] 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. [226] mention that asynchronous domain decomposition methods are by definition fault-tolerant. In [227–229], 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 [117], 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 [230]. 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. Finally, 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. We refer the reader to the recently published report by Radojkovic et al. [7] for an overview of the needs of the next generation HPC systems.

4.1.1 Error correcting codes Poulos et al. [231] 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. For the application studied in [231], 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 Checkpoint/restart will remain a necessary component for any future resilient 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, where 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.

- Localize checkpoint data to own or localized nodes. This could be supported by local non-volatile memory, as targets for checkpoint data. While this has the potential to reduce communication, as it avoids remote data transfers, it may require additional hardware support to retrieve data from non-functional nodes, e.g., by accessing data through fast JTAG-like interfaces.
- In memory checkpointing.
- Exploit user-level knowledge for serializing, packing, compressing data, see e.g., how existing AMR functionality [57] can be exploited for efficient checkpointing in Section. 2.2.1.

Checkpoint/Restart for heterogeneous systems In addition to classical checkpoint/restart for homogeneous systems, node-local checkpoint/restart support for heterogeneous systems will help containing error and failure propagation. Such support may be provided transparently to the application by the underlying infrastructure, such as GPU drivers or task-based environments, or exposed in the programming model, such as OpenMP Offload [232].

4.1.3 Scheduler and resource management

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

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

Adaptive system and application load balancing The batch scheduler needs to adaptively balance the system load onto the available resources, via seamless application migration. While the application needs to adapt to the capabilities of the newly allocated resources, if different from the original allocation, without incurring performance penalties. The former has typically been implemented via checkpointing and process migration [233]. The latter has typically been implemented for applications that can adjust their granularity, e.g., from finer to coarser, depending on resource availability either triggered by the application or the system [234]. When exposing and expressing parallelism in applications, in addition to accounting for and matching the multiple levels of hardware parallelism (nodes, sockets, cores), the decomposition granularity needs to be flexible to support evolvability and malleability and allow for adaptive load balancing at the application and system levels.

4.2 Programming models with inherent resiliency support

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

Table 2. Properties of numerical algorithms fostering or helping resilience

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

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 Resilience friendly algorithms

The metrics used to measure the efficiency of an algorithm have changed with evolving architectures. Historically, algorithms were ranked according to their flops. More recently, the metrics were expanded to acknowledge memory access patterns. In this section we mention some of the types of algorithms that are likely to be deemed efficient if the metrics are modified to include resilience. 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. Table 2 lists properties that we found are, or can be, contributing to the resilience of these algorithms.

4.3.1 Iterative methods Fundamentally, iterative solvers may be viewed as inherently more robust than direct solvers because they do not compute their solution using a predefined sequence of numerical operations as direct solvers typically do. A number of examples of iterative methods have been given in Section 3.

4.3.2 Locality, asynchronicity and embarrassing 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 Element Method* (FEM)), finite differences and finite volumes. See Section 3.2.2.

Domain decomposition schemes such as additive Schwarz methods, substructuring-inspired FETI [235], the fully adaptive multigrid method [214], and the stochastic subspace correction method [225] are naturally asynchronous and resilient to message loss. Note that the matematical properties of such asynchronous methods contradict the

requirements of classical Krylov methods when used as preconditioners. In this case, Krylov acceleration must be used in a flexible form.

Note that here, we use the term asynchronous primarily in the sense of reducing the time synchronicity in parallel computations. Section 3.2.1 included a number of examples of these types of algorithms. Both the localized and asynchronous approaches, achieve their impact through a decoupling of computations. Going further in this direction may lead to embarrassingly or nearly embarrassingly parallel algorithms. Examples of such methods are Monte Carlo simulations which are discussed in the next section.

4.3.3 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. 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.

The idea of interval arithmetic is to compute bounds of intervals that always contain the exact result [236, 237]. Probabilistic methods for rounding error estimation [238–242] require several executions of arithmetic operations with different perturbations or different rounding modes (for instance three executions for Discrete Stochastic Arithmetic [243]). 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 Redundancy and replication 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.

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. 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 previous subsection on stochastic methods.

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.

4.3.5 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 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).

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. As stated in Section 4.3.4, 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.

4.3.6 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 [173, 244–249], 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 [250–256], which are however still far from mainstream for most applications. For PDE solvers, local spatial adaptivity techniques are also very common [257, 258], but their incorporation in operational applications is still a research topic, see e.g. [259–263] 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 error detection/DeB strategies, for example activating an additional restart of the Arnoldi procedure as soon as an increase in the residual norm is observed.

4.3.7 Low memory footprint – matrix-free The classical approach to represent linear operators as sparse matrices produces large amounts of static data which has to be restored upon failure. Matrix-free methods do not represent the operators as static data. Therefore, large sparse matrix data structures do not have to be restored upon failure as they are computed on the fly in any event. Extreme-scale applications will benefit from matrix-free approaches due to their low memory footprint, also in terms of runtime, (due to high memory access cost) and higher limits for the overall problem size [264, 265].

In addition to saving memory and, therewith, reducing the risk of memory corruption, matrix-free methods can also be combined with automatic code generation [266] in a stencil-based approach, i.e., for finite difference methods on uniform structured grids. In such cases, the matrix entries may be 'hard wired' into code, such as 5-point stencils for Laplace's equation. Automatic code generation provides a means to increase resiliency in the code generator or domain specific language and, thus, facilitate resilience aware software development.

For FEM, one can use local assembly kernels [267]. Here, the trade-off between computation and storage and, in the future, resilience is relevant in particular for higher order elements.

4.4 The final mile: towards a resilient ecosystem

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

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

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

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

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

Tools for validation Currently, there are no standard tools to test the correctness and performance of resilient algorithms under undetected errors and fault. This is due to a lack of fault injection tools that reflect realistic situations. Debardeleben et al. [268] have developed a hardware error simulator tool to understand the behavior of numerical algorithms under faulty hardware with a great accuracy, but this approach cannot evaluate the execution time of resilient algorithms at scale. Vendors provide fault injection tools [269, 270] for better execution efficiency, compromising the accuracy of the hardware behavior. Compiler approaches or other in-house error injections [271, 272] could allow the program to execute as efficiently as the original binary, but the correctness is further compromised. There are also tools that can analyze an application's vulnerability very quickly but do not actually produce the application's faulty output. One technique for this, DiSCvar [273], uses algorithmic differentiation and exposes how changes to each variable impact output results. It is important to note that these techniques do not actually produce that corrupted output. Hence, they are very fast but they may not be useful to developers looking to explore precisely how corruption changes their application. It is likely that a combination of these techniques, which identify most critical regions of an application coupled with fault injection at those locations, may serve as a good compromise between the two techniques.

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

Tools for code transformation Compilers are able to generate binaries with resilience capability as suggested in the work by [274]; the generated binary instruments redundant computation, register allocations to enable error detection and correction during program execution. The recent work by Lin [275] leverages LLVM to generate SIMD

instructions to perform redundant computation and verification. Source-to-source code transformation has been proposed to enable triple modular redundancy in loops [276] and automatic instrumentation of checkpointing [277]. Similarly, this idea can be extended to redundant threading for error mitigation, facilitated with OpenMP-like programming language extension [278]. These approaches automatically introduce resilience with some performance penalty, preventing the users from selective adaptation of resilience for performance optimization, and these redundant computations are benefited from the memory hierarchy, preventing doubling (or tripling) of the execution time.

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.

Acknowledgements

We would like to thanks the anonymous referees for their valuable remarks and comments that have contributed to substantially improve this paper.

Declaration of conflicting interests

The Authors declare that there is no conflict of interest.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

[1] Herault T and Robert Y (eds.) *Fault-Tolerance Techniques for High-Performance Computing*. Springer Verlag, 2015.

[2] von Neumann J. First Draft of a Report on the EDVAC, 1945. URL https://nsu.ru/xmlui/bitstream/handle/nsu/9018/2003-08-TheFirstDraft.pdf?sequence=1&isAllowed=y.

[3] Cappello F, Geist A, Gropp B et al. Toward exascale resilience. *The International Journal of High Performance Computing Applications* 2009; 23(4): 374–388.

[4] Avizienis A, Laprie J, Randell B et al. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing* 2004; 1(1): 11–33. DOI:10.1109/TDSC.2004.2.

[5] Benacchio T, Bonaventura L, Altenbernd M et al. Report on local data recovery approaches suitable for weather and climate prediction. FET-HPC ESCAPE-2 project deliverable, 2020. DOI:10.2172/1607968.

[6] Benacchio T, Bonaventura L, Altenbernd M et al. Resilience and fault-tolerance in high-performance computing for numerical weather and climate prediction. *International Journal of High Performance Computing Applications* 2021; DOI: 10.1177/1094342021990433.

[7] Radojkovic P, Marazakis M, Carpenter P et al. Towards Resilient EU HPC Systems: A Blueprint. Research report, European HPC resilience initiative, 2020. URL https://hal.inria.fr/hal-02922257.

[8] Fiala D, Mueller F, Ferreira K et al. Mini-Ckpts: Surviving OS failures in persistent memory. In *Proceedings of the* 30^{th} *ACM International Conference on Supercomputing (ICS) 2016*. Istanbul, Turkey: ACM Press, New York, NY, USA. ISBN 978-1-4503-4361-9, pp. 7:1–7:14. DOI:10.1145/2925426.2926295.

[9] Hargrove PH and Duell JC. 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. Denver, CO, USA: Institute of Physics Publishing, Bristol, UK, pp. 494–499. DOI: 10.1088/1742-6596/46/1/067.

[10] Forum M. The Message Passing Interface standard, 2019. URL `https://www.mpi-forum.org/`.

[11] Varma J, Wang C, Mueller F et al. Scalable, fault-tolerant membership for MPI tasks on HPC systems. In *Proceedings of the* 20^{th} *ACM International Conference on Supercomputing (ICS) 2006*. Cairns, Australia: ACM Press, New York, NY, USA. ISBN 1-59593-282-8, pp. 219–228. DOI:10.1145/1183401.1183433.

[12] Wang C, Mueller F, Engelmann C et al. A job pause service under LAM/MPI+BLCR for transparent fault tolerance. In *Proceedings of the* 21^{st} *IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2007*. Long Beach, CA, USA: ACM Press, New York, NY, USA. ISBN 978-1-59593-768-1, pp. 1–10. DOI:10.1109/IPDPS.2007.370307.

[13] Wang C, Mueller F, Engelmann C et al. Hybrid checkpointing for MPI jobs in HPC environments. In *Proceedings of the* 16^{th} *IEEE International Conference on Parallel and Distributed Systems (ICPADS) 2010*. Shanghai, China: IEEE Computer Society, Los Alamitos, CA, USA. ISBN 978-0-7695-4307-9, pp. 524–533. DOI:10.1109/ICPADS.2010.48.

[14] Gioiosa R, Sancho JC, Jiang S et al. 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*. IEEE, pp. 9–9.

[15] Fiala D, Mueller F and Ferreira KB. Flipsphere: A software-based DRAM error detection and correction library for HPC. In *Proceedings of the* 20^{th} *IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT) 2016*. London, UK: IEEE Computer Society, Los Alamitos, CA, USA. ISBN 978-1-5090-3506-9, pp. 19–28. DOI:10.1109/DS-RT.2016.27.

[16] Di S and Cappello F. Adaptive impact-driven detection of silent data corruption for hpc applications. *IEEE Transactions on Parallel and Distributed Systems* 2016; 27(10): 2809–2823. DOI:10.1109/TPDS.2016.2517639.

[17] Subasi O, Di S, Balaprakash P et al. Macord: Online adaptive machine learning framework for silent error detection. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. pp. 717–724. DOI:10.1109/CLUSTER.2017.128.

[18] Li S, Di S, Zhao K et al. Towards end-to-end sdc detection for hpc applications equipped with lossy compression. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. pp. 326–336. DOI:10.1109/CLUSTER49012.2020.00043.

[19] Engelmann C, Vallée GR, Naughton T et al. Proactive fault tolerance using preemptive migration. In *Proceedings of the* 17^{th} *Euromicro International Conference on Parallel, Distributed, and network-based Processing (PDP) 2009*. Weimar, Germany: IEEE Computer Society, Los Alamitos, CA, USA. ISBN 978-0-7695-3544-9, pp. 252–257. DOI:10.1109/PDP.2009.31.

[20] Wang C, Mueller F, Engelmann C et al. Proactive process-level live migration and back migration in HPC environments. *Journal of Parallel and Distributed Computing (JPDC)* 2012; 72(2): 254–267. DOI:10.1016/j.jpdc.2011.10.009.

[21] Nagarajan AB, Mueller F, Engelmann C et al. Proactive fault tolerance for HPC with Xen virtualization. In *Proceedings of the* 21^{st} *ACM International Conference on Supercomputing (ICS) 2007*. Seattle, WA, USA: ACM Press, New York, NY, USA. ISBN 978-1-59593-768-1, pp. 23–32. DOI:10.1145/1274971.1274978.

[22] Mattson T, Cledat R, Cave V et al. The open community runtime: A runtime system for extreme scale computing. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. pp. 1–7. DOI:10.1109/HPEC.2016.7761580.

[23] Thibault S. *On Runtime Systems for Task-based Programming on Heterogeneous Platforms*. Habilitation à diriger des recherches, Université de Bordeaux, 2018. URL `https://hal.inria.fr/tel-01959127`.

[24] Lion R and Thibault S. From tasks graphs to asynchronous distributed checkpointing with local restart. In *Fault Tolerance for HPC at eXtreme Scale (FTXS) Workshop*.

[25] Lion R. Tolérance aux pannes dans l'exécution distribuée de graphes de tâches. In *Conférence d'informatique en Parallélisme, Architecture et Système*. Anglet, France. URL `https://hal.inria.fr/hal-02296118`.

[26] Subasi O, Arias J, Unsal O et al. Nanocheckpoints: A task-based asynchronous dataflow framework for efficient and scalable checkpoint/restart. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. pp. 99–102.

[27] Subasi O, Yalcin G, Zyulkyarov F et al. A runtime heuristic to selectively replicate tasks for application-specific reliability targets. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. pp. 498–505.

[28] Paul SR, Hayashi A, Slattengren N et al. Enabling resilience in asynchronous many-task programming models. In Yahyapour R (ed.) *Euro-Par 2019: Parallel*

Processing. Springer International Publishing. ISBN 978-3-030-29400-7, pp. 346–360.

[29] Kale LV and Krishnan S. Charm++: A portable concurrent object oriented system based on C++. In *Proceedings of the Eighth Annual Conference on Object-Oriented Programming Systems, Languages, and Applications*. OOPSLA'93, New York, NY, USA: Association for Computing Machinery. ISBN 0897915879, p. 91–108. DOI:10.1145/165854.165874. URL https://doi.org/10.1145/165854.165874.

[30] Yan Y, Zhao J, Guo Y et al. Hierarchical place trees: A portable abstraction for task parallelism and data movement. In *International Workshop on Languages and Compilers for Parallel Computing*. Springer, pp. 172–187.

[31] Kaiser H, Heller T, Adelstein-Lelbach B et al. HPX: A task based programming model in a global address space. In *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*. PGAS'14, New York, NY, USA: Association for Computing Machinery. ISBN 9781450332477. DOI:10.1145/2676870.2676883. URL https://doi.org/10.1145/2676870.2676883.

[32] Duran A, Ayguadé E, Badia RM et al. OmpSs: a proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters* 2011; 21(2): 173–193. URL http://dblp.uni-trier.de/db/journals/ppl/ppl21.html#DuranABLMMP11.

[33] Pena AJ, Bland W and Balaji P. VOCL-FT: Introducing techniques for efficient soft error coprocessor recovery. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'15)*. pp. 1–12. DOI:10.1145/2807591.2807640.

[34] Group TK. OpenCL (Open Computing Language), 2020. URL https://www.khronos.org/opencl/.

[35] Boards TOAR. OpenMP (Open Multi-Processing), 2019. URL https://www.openmp.org/.

[36] Engelmann C, Vallée GR and Pophale S. Concepts for OpenMP target offload resilience. In *Lecture Notes in Computer Science: Proceedings of the* 15^{th} *International Workshop on OpenMP (IWOMP) 2019*, volume 11718. Auckland, New Zealand: Springer Verlag, Berlin, Germany. ISBN 978-3-030-28595-1, pp. 78–93. DOI:10.1007/978-3-030-28596-8_6.

[37] Casanova H, Robert Y, Vivien F et al. On the impact of process replication on executions of large-scale parallel applications with coordinated checkpointing. *Future Generation Comp Syst* 2015; 51: 7–19.

[38] Yu J, Jian D, Wu Z et al. Thread-level redundancy fault tolerant CMP based on relaxed input replication. In *2011 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*. pp. 544–549.

[39] Crago SP, Kang DI, Kang M et al. Programming models and development software for a space-based many-core processor. In *4th Int. Conf. on Space Mission Challenges for Information Technology*. IEEE, pp. 95–102.

[40] Rashid MW and Huang MC. Supporting highly-decoupled thread-level redundancy for parallel programs. In *Conf. on High-Performance Computer Architecture (HPCA)*. IEEE, pp. 393–404.

[41] Cluster File Systems, Inc. Lustre: A scalable, high-performance file system. White paper, Available at `https://cse.buffalo.edu/faculty/tkosar/cse710/papers/lustre-whitepaper.pdf`, 2007.

[42] He XB, Ou L, Engelmann C et al. Symmetric active/active metadata service for high availability parallel file systems. *Journal of Parallel and Distributed Computing (JPDC)* 2009; 69(12): 961–973. DOI:10.1016/j.jpdc.2009.08.004.

[43] Elliott J, Kharbas K, Fiala D et al. Combining partial redundancy and checkpointing for HPC. In *2012 IEEE 32nd International Conference on Distributed Computing Systems (ICDCS)*. pp. 615–626. DOI:10.1109/ICDCS.2012.56.

[44] Stearley J, Ferreira K, Robinson D et al. Does partial replication pay off? In *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)*. pp. 1–6. DOI:10.1109/DSNW.2012.6264669.

[45] Subasi O, Arias J, Unsal O et al. Programmer-directed partial redundancy for resilient HPC. In *Proceedings of the 12th ACM International Conference on Computing Frontiers (CF)*. New York, NY, USA: ACM. ISBN 978-1-4503-3358-0, pp. 47:1–47:2. DOI:10.1145/2742854.2742903. URL `http://doi.acm.org/10.1145/2742854.2742903`.

[46] Subasi O, Yalcin G, Zyulkyarov F et al. Designing and modelling selective replication for fault-tolerant HPC applications. In *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. pp. 452–457. DOI:10.1109/CCGRID.2017.40.

[47] George C and Vadhiyar SS. ADFT: An adaptive framework for fault tolerance on large scale systems using application malleability. *Procedia Computer Science* 2012; 9: 166 – 175.

[48] Berrocal E, Bautista-Gomez L, Di S et al. Exploring partial replication to improve lightweight silent data corruption detection for HPC applications. In *Euro-Par 2016: Parallel Processing - 22nd International Conference on Parallel and*

Distributed Computing, Grenoble, France, August 24-26, 2016, Proceedings. pp. 419–430. DOI:10.1007/978-3-319-43659-3_31. URL https://doi.org/10.1007/978-3-319-43659-3_31.

[49] Berrocal E, Bautista-Gomez L, Di S et al. Toward general software level silent data corruption detection for parallel applications. *IEEE Trans Parallel Distrib Syst* 2017; 28(12): 3642–3655. DOI:10.1109/TPDS.2017.2735971. URL https://doi.org/10.1109/TPDS.2017.2735971.

[50] Ni X and Kale LV. FlipBack: Automatic targeted protection against silent data corruption. In *Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. pp. 335–346. DOI:10.1109/SC.2016.28.

[51] Cavelan A, Cabezón RM and Ciorba FM. Detection of silent data corruptions in smoothed particle hydrodynamics simulations. In *Proceedings of the 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2019)*.

[52] Fiala D, Mueller F, Engelmann C et al. Detection and correction of silent data corruption for large-scale high-performance computing. In *Proceedings of the* 25^{th} *IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC) 2012*. Salt Lake City, UT, USA: ACM Press, New York, NY, USA. ISBN 978-1-4673-0804-5, pp. 78:1–78:12. DOI: 10.1109/SC.2012.49.

[53] Levy S, Topp B, Ferreira KB et al. 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*. Springer, pp. 91–114.

[54] Engelmann C and Naughton T. Toward a performance/resilience tool for hardware/software co-design of high-performance computing systems. In *2013 42nd International Conference on Parallel Processing*. IEEE, pp. 960–969.

[55] Ashraf RA and Engelmann C. Performance efficient multiresilience using checkpoint recovery in iterative algorithms. In *European Conference on Parallel Processing*. Springer, pp. 813–825.

[56] Di S, Bouguerra MS, Bautista-Gomez L et al. Optimization of multi-level checkpoint model for large scale HPC applications. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, pp. 1181–1190.

[57] Kohl N, Hötzer J, Schornbaum F et al. A scalable and extensible checkpointing scheme for massively parallel simulations. *The International Journal of High Performance Computing Applications* 2019; 33(4): 571–589.

[58] Zheng G, Ni X and Kalé LV. A scalable double in-memory checkpoint and restart scheme towards exascale. In *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)*. IEEE, pp. 1–6.

[59] Bautista-Gomez L, Tsuboi S, Komatitsch D et al. FTI: high performance fault tolerance interface for hybrid systems. In *Proceedings of 2011 international conference for high performance computing, networking, storage and analysis*. pp. 1–32.

[60] Benoit A, Cavelan A, Robert Y et al. Optimal resilience patterns to cope with fail-stop and silent errors. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, pp. 202–211.

[61] Coti C, Herault T, Lemarinier P et al. Blocking vs. non-blocking coordinated checkpointing for large-scale fault tolerant MPI. In *SC'06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. IEEE, pp. 18–18.

[62] Sato K, Maruyama N, Mohror K et al. Design and modeling of a non-blocking checkpointing system. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, pp. 1–10.

[63] Di S and Cappello F. Fast error-bounded lossy HPC data compression with SZ. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, pp. 730–739.

[64] Kim J, Lee S and Vetter JS. PapyrusKV: A high-performance parallel key-value store for distributed NVM architectures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. New York, NY, USA: Association for Computing Machinery. ISBN 9781450351140. DOI:10.1145/3126908.3126943. URL `https://doi.org/10.1145/3126908.3126943`.

[65] Tang X, Zhai J, Yu B et al. An efficient in-memory checkpoint method and its practice on fault-tolerant HPL. *IEEE Transactions on Parallel and Distributed Systems* 2018; 29(4): 758–771. DOI:10.1109/TPDS.2017.2781257.

[66] Moody A, Bronevetsky G, Mohror K et al. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. pp. 1–11. DOI:10.1109/SC.2010.18.

[67] Keller K and Bautista-Gomez L. Application-level differential checkpointing for HPC applications with dynamic datasets. In *19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2019, Larnaca, Cyprus, May 14-17, 2019*. IEEE, pp. 52–61. DOI:10.1109/CCGRID.2019.00015. URL `https://doi.org/10.1109/CCGRID.2019.00015`.

[68] Cantwell CD and Nielsen AS. A minimally intrusive low-memory approach to resilience for existing transient solvers. *Journal of Scientific Computing* 2019; 78(1): 565–581.

[69] Ropars T, Martsinkevich TV, Guermouche A et al. SPBC: Leveraging the characteristics of MPI HPC applications for scalable checkpointing. In *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. pp. 1–12. DOI:10.1145/2503210.2503271.

[70] Subasi O, Martsinkevich T, Zyulkyarov F et al. Unified fault-tolerance framework for hybrid task-parallel message-passing applications. *The International Journal of High Performance Computing Applications* 2018; 32(5): 641–657.

[71] Losada N, Bosilca G, Bouteiller A et al. Local rollback for resilient MPI applications with application-level checkpointing and message logging. *Future Generation Computer Systems* 2019; 91: 450–464.

[72] Moody A, Bronevetsky G, Mohror K et al. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '10, USA: IEEE Computer Society. ISBN 9781424475599, p. 1–11. DOI:10.1109/SC.2010.18. URL `https://doi.org/10.1109/SC.2010.18`.

[73] Shahzad F, Thies J, Kreutzer M et al. CRAFT: A library for easier application-level checkpoint/restart and automatic fault tolerance. *IEEE Transactions on Parallel and Distributed Systems* 2018; 30(3): 501–514. DOI:10.1109/TPDS.2018.2866794.

[74] Nicolae B, Moody A, Gonsiorowski E et al. VeloC: Towards high performance adaptive asynchronous checkpointing at large scale. In *IPDPS'19: The 2019 IEEE International Parallel and Distributed Processing Symposium*. Rio de Janeiro, Brazil, pp. 911–920. URL `https://hal.archives-ouvertes.fr/hal-02184203`.

[75] Team CD. Containment domains, 2014. URL `https://lph.ece.utexas.edu/public/CDs/ContainmentDomains`.

[76] Austin B, Roman E and Li X. Resilient matrix multiplication of hierarchical semi-separable matrices. In *Proceedings of the 5th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS)*. Portland, Oregon, pp. 19–26. DOI:10.1145/2751504.2751507.

[77] Chien AA, Balaji P, Dun N et al. Exploring versioned distributed arrays for resilience in scientific applications. *IJHPCA* 2017; 31(6): 564–590. DOI:10.1177/1094342016664796. URL `https://doi.org/10.1177/1094342016664796`.

[78] Zheng Z, Chien AA and Teranishi K. Fault tolerance in an inner-outer solver: A GVR-enabled case study. In Daydé MJ, Marques O and Nakajima K (eds.) *High Performance Computing for Computational Science - VECPAR 2014 - 11th International Conference, Eugene, OR, USA, June 30 - July 3, 2014, Revised Selected Papers*, *Lecture Notes in Computer Science*, volume 8969. Springer, pp. 124–132. DOI:10.1007/978-3-319-17353-5_11. URL https://doi.org/10.1007/978-3-319-17353-5_11.

[79] Team TZ. Zoltan: Parallel Partitioning, Load Balancing and Data-Management Services, 2013. URL https://cs.sandia.gov/Zoltan/.

[80] Carino RL, Mohammed A and Ciorba FM. Dynamic Loop Self-scheduling For Load Balancing (DLS4LB), 2020. URL https://github.com/unibas-dmi-hpc/DLS4LB.

[81] Team TM. MERCURY programming language, 2020. URL https://www.mercurylang.org/.

[82] team N. Nanos++, 2020. URL https://www.bsc.es/research-and-development/software-and-apps/software-list/nanos-rtl.

[83] Hassani A, Skjellum A and Brightwell R. 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*. IEEE Computer Society, pp. 750–755. DOI: 10.1109/DSN.2014.78. URL https://doi.org/10.1109/DSN.2014.78.

[84] Bland W, Bouteiller A, Herault T et al. An evaluation of user-level failure mitigation support in MPI. In *Proceedings of the 19th European Conference on Recent Advances in the Message Passing Interface*. EuroMPI'12, Berlin, Heidelberg: Springer-Verlag. ISBN 978-3-642-33517-4, pp. 193–203. DOI: 10.1007/978-3-642-33518-1_24.

[85] Laguna I, Richards D, Gamblin T et al. Evaluating and extending user-level fault tolerance in MPI applications. *The International Journal of High Performance Computing Applications* 2016; 30. DOI:10.1177/1094342015623623.

[86] Gamell M, Katz DS, Kolla H et al. 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*. pp. 895–906. DOI:10.1109/SC.2014.78.

[87] Gamell M, Katz DS, Teranishi K et al. Evaluating online global recovery with Fenix using application-aware in-memory checkpointing techniques. In *ICPP Workshops*. IEEE Computer Society. ISBN 978-1-5090-2825-2,

pp. 346–355. URL http://dblp.uni-trier.de/db/conf/icppw/icppw2016.html#GamellKTHWMP16.

[88] Teranishi K and Heroux MA. Toward local failure local recovery resilience model using MPI-ULFM. In *Proceedings of the 21st European MPI Users' Group Meeting*. EuroMPI/ASIA'14, New York, NY, USA: Association for Computing Machinery. ISBN 9781450328753, p. 51–56. DOI:10.1145/2642769.2642774. URL https://doi.org/10.1145/2642769.2642774.

[89] Shahzad F, Thies J, Kreutzer M et al. CRAFT: A library for easier application-level checkpoint/restart and automatic fault tolerance. *IEEE Transactions on Parallel and Distributed Systems* 2019; 30(3): 501–514. DOI:10.1109/TPDS.2018.2866794.

[90] Sukhija N, Banicescu I and Ciorba FM. Investigating the resilience of dynamic loop scheduling in heterogeneous computing systems. In *Proceedings of the 14th International Symposium on Parallel and Distributed Computing (ISPDC 2015)*. pp. 194–203. DOI:10.1109/ISPDC.2015.29.

[91] Mohammed A, Cavelan A and Ciorba FM. rDLB: A novel approach for robust dynamic load balancing of scientific applications with independent tasks. In *Proceedings of the 2019 International Conference on High Performance Computing & Simulation (HPCS 2019)*. Dublin.

[92] Edwards HC, Trott CR and Sunderland D. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing* 2014; 74(12): 3202 – 3216. DOI:https://doi.org/10.1016/j.jpdc.2014.07.003. URL http://www.sciencedirect.com/science/article/pii/S0743731514001257. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[93] Beckingsale DA, Burmark J, Hornung R et al. RAJA: Portable performance for large-scale scientific applications. In *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*. pp. 71–81. DOI:10.1109/P3HPC49587.2019.00012.

[94] team R. Raja performance portability layer. https://github.com/LLNL/RAJA, 2019.

[95] Miles JS, Teranishi K, Morales NM et al. Software resilience using kokkos ecosystem. Technical Report SAND2019-3616, Sandia National Laboratory, Albuquerque, New Mexico, 2019.

[96] Hukerikar S and Engelmann C. Resilience design patterns: A structured approach to resilience at extreme scale (version 1.2). Technical Report ORNL/TM-2017/745, Oak Ridge National Laboratory, Oak Ridge, TN, USA, 2017. DOI:10.2172/1436045.

[97] Hukerikar S and Engelmann C. A pattern language for high-performance computing resilience. In *Proceedings of the* 22^{nd} *European Conference on Pattern Languages of Programs (EuroPLoP) 2017*. Kloster Irsee, Germany: ACM Press, New York, NY, USA. ISBN 978-1-4503-4848-5, pp. 12:1–12:16. DOI: 10.1145/3147704.3147718.

[98] Ashraf R, Hukerikar S and Engelmann C. Pattern-based modeling of multiresilience solutions for high-performance computing. In *Proceedings of the* 9^{th} *ACM/SPEC International Conference on Performance Engineering (ICPE) 2018*. Berlin, Germany: ACM Press, New York, NY, USA. ISBN 978-1-4503-5095-2, pp. 80–87. DOI:10.1145/3184407.3184421.

[99] Huang Kh and Abraham Ja. Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on Computers* 1984; c(6): 518–528. DOI:10.1109/TC.1984.1676475.

[100] Wu P and Chen Z. FT-ScaLAPACK: Correcting soft errors on-line for ScaLAPACK Cholesky, QR, and LU factorization routines. In *Proceedings of the 23rd International Symposium on High-per formance Parallel and Distributed Computing*. HPDC '14, New York, NY, USA: ACM. ISBN 978-1-4503-2749-7, pp. 49–60. DOI:10.1145/2600212.2600232. URL `http://doi.acm.org/10.1145/2600212.2600232`.

[101] Wu P, Guan Q, DeBardeleben N et al. Towards practical algorithm based fault tolerance in dense linear algebra. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*. New York, NY, USA: Association for Computing Machinery. ISBN 9781450343145. DOI:10.1145/2907294.2907315. URL `https://doi.org/10.1145/2907294.2907315`. Dense linear algebra.

[102] Mishra A and Banerjee P. An algorithm-based error detection scheme for the multigrid method. *IEEE Trans Comput* 2003; 52(9). DOI:10.1109/TC.2003.1228507. URL `https://doi.org/10.1109/TC.2003.1228507`.

[103] Tao D, Song SL, Krishnamoorthy S et al. New-sum: A novel online ABFT scheme for general iterative methods. In Nakashima H, Taura K and Lange J (eds.) *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing, HPDC 2016, Kyoto, Japan, May 31 - June 04, 2016*. ACM, pp. 43–55. DOI:10.1145/2907294.2907306. URL `https://doi.org/10.1145/2907294.2907306`. Iterative methods for linear systems: Krylov, stationary.

[104] Shantharam M, Srinivasmurthy S and Raghavan P. Characterizing the impact of soft errors on iterative methods in scientific computing. *Proceedings of the international conference on Supercomputing - ICS '11* 2011; : 152DOI: 10.1145/1995896.1995922. URL `http://portal.acm.org/citation.cfm?doid=1995896.1995922`.

[105] Agullo E, Cools S, Fatih-Yetkin E et al. On soft errors in the Conjugate Gradient method: sensitivity and robust numerical detection - revised. Research Report RR-9330, Inria, 2020. URL https://hal.inria.fr/hal-02495301.

[106] Liang X, Chen J, Tao D et al. 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: Association for Computing Machinery. ISBN 9781450351140. DOI:10.1145/3126908.3126915. URL https://doi.org/10.1145/3126908.3126915.

[107] Zhao K, Di S, Li S et al. FT-CNN: Algorithm-based fault tolerance for convolutional neural networks. *IEEE Transactions on Parallel and Distributed Systems* 2021; : 1–1DOI:10.1109/tpds.2020.3043449. URL http://dx.doi.org/10.1109/TPDS.2020.3043449.

[108] Hestenes MR and Stiefel E. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards* 1952; 46(6): 409–436.

[109] Mycek P, Rizzi F, Maître OL et al. Discrete a priori bounds for the detection of corrupted PDE solutions in exascale computations. *SIAM Journal on Scientific Computing* 2017; 39(1): C1–C28. DOI:10.1137/15M1051786. URL https://doi.org/10.1137/15M1051786. https://doi.org/10.1137/15M1051786.

[110] Higham N. *Accuracy and Stability of Numerical Algorithms*. 1st ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1996. ISBN 0898713552.

[111] Liesen J and Strakoš Z. *Krylov Subspace Methods*. Numerical Mathematics and Scientific Computation, Oxford University Press, 2013. ISBN 978-0-19-965541-0.

[112] Meurant G and Strakoš Z. The Lanczos and Conjugate Gradient algorithms in finite precision arithmetic. *Acta Numerica* 2006; 15: 471–542.

[113] van der Vorst HA and Ye Q. Residual replacement strategies for Krylov subspace iterative methods for the convergence of true residuals. *SIAM Journal on Scientific Computing* 2000; 22(3): 835–852.

[114] Chen Z. Online-ABFT: An online algorithm based fault tolerance scheme for soft error detection in iterative methods. *SIGPLAN Not* 2013; 48(8). DOI:10.1145/2517327.2442533. URL https://doi.org/10.1145/2517327.2442533.

[115] Calhoun J, Olson L, Snir M et al. Towards a more fault resilient multigrid solver. In *Proceedings of the Symposium on High Performance Computing*. San Diego, CA, USA: Society for Computer Simulation International. ISBN 9781510801011.

[116] Altenbernd M and Göddeke D. Soft fault detection and correction for multigrid. *The International Journal of High Performance Computing Applications* 2018; 32(6): 897–912. DOI:10.1177/1094342016684006.

[117] Grout R, Kolla H, Minion M et al. Achieving algorithmic resilience for temporal integration through spectral deferred corrections. *Communications in Applied Mathematics and Computational Science* 2017; 12(1): 25–50.

[118] Nielsen AS and Hesthaven JS. Fault tolerance in the Parareal method. In *Proceedings of the ACM Workshop on Fault-Tolerance for HPC at Extreme Scale*. FTXS '16, New York, NY, USA: ACM. ISBN 978-1-4503-4349-7, pp. 1–8. DOI:10.1145/2909428.2909431. URL `http://dx.doi.org/10.1145/2909428.2909431`.

[119] Benson AR, Schmit S and Schreiber R. Silent error detection in numerical time-stepping schemes. *IJHPCA* 2015; 29(4): 403–421. DOI:10.1177/1094342014532297. URL `https://doi.org/10.1177/1094342014532297`. `https://doi.org/10.1177/1094342014532297`.

[120] Guhur PL, Zhang H, Peterka T et al. Lightweight and accurate silent data corruption detection in ordinary differential equation solvers. In *European Conference on Parallel Processing*. Springer, pp. 644–656.

[121] Bungartz HJ and Griebel M. Sparse grids. *Acta Numerica* 2004; 13: 147–269. DOI:10.1017/S0962492904000182.

[122] Jakeman JD and Roberts SG. Local and dimension adaptive sparse grid interpolation and quadrature. *arXiv preprint arXiv:11100010* 2011; September. `1110.0010v1`.

[123] Gerstner T and Griebel M. Dimension–adaptive tensor–product quadrature. *Computing* 2003; 71(1): 65–87. DOI:10.1007/s00607-003-0015-5.

[124] Gerstner T and Griebel M. Numerical integration using sparse grids. *Numerical Algorithms* 1998; 18(3): 209. DOI:10.1023/A:1019129717644.

[125] Bungartz HJ and Dirnstorfer S. Multivariate quadrature on adaptive sparse grids. *Computing* 2003; 71(1): 89–114. DOI:10.1007/s00607-003-0016-4.

[126] Heene M, Hinojosa AP, Obersteiner M et al. 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*. Springer, 2018. pp. 513–529.

[127] Harding B and Hegland M. Robust solutions to PDEs with multiple grids. In Garcke J and Pflüger D (eds.) *Sparse Grids and Applications - Munich 2012 SE*, *Lecture Notes in Computational Science and Engineering*, volume 97. Springer International Publishing. ISBN 978-3-319-04536-8, 2014. pp. 171–193.

[128] Garcke J. Regression with the optimised combination technique. In *Proceedings of the 23rd international conference on Machine learning*. ACM Press, pp. 321–328.

[129] Garcke J. A dimension adaptive sparse grid combination technique for machine learning. *ANZIAM Journal* 2007; 48: 725–740.

[130] Peherstorfer B, Pflüge D and Bungartz HJ. Density estimation with adaptive sparse grids for large data sets. In *Proceedings of the 2014 SIAM international conference on data mining*. SIAM, pp. 443–451. DOI:10.1137/1.9781611973440.51. URL `https://epubs.siam.org/doi/abs/10.1137/1.9781611973440.51`. `https://epubs.siam.org/doi/pdf/10.1137/1.9781611973440.51`.

[131] Griebel M, Schneider M and Zenger C. A combination technique for the solution of sparse grid problems. In *Iterative Methods in Lin. Alg.* pp. 263–281.

[132] Obersteiner M, Hinojosa AP, Heene M et al. A highly scalable, algorithm-based fault-tolerant solver for gyrokinetic plasma simulations. In *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*. pp. 1–8.

[133] Ali MM, Strazdins PE, Harding B et al. A fault-tolerant gyrokinetic plasma application using the sparse grid combination technique. In *Proceedings of the 2015 International Conference on High Performance Computing & Simulation (HPCS 2015)*. Amsterdam, The Netherlands, pp. 499–507.

[134] Ali MM, Strazdins PE, Harding B et al. Complex scientific applications made fault-tolerant with the sparse grid combination technique. *International Journal of High Performance Computing Applications* 2016; 30(3): 335–359.

[135] Heene M, Parra Hinojosa A, Bungartz HJ et al. A massively-parallel, fault-tolerant solver for high-dimensional PDEs. In *Euro-Par 2016: Parallel Processing Workshops*.

[136] Harding B et al. Fault tolerant computation with the sparse grid combination technique. *SIAM Journal on Scient Comp* 2015; 37(3): C331–C353.

[137] Hinojosa AP, Harding B, Hegland M et al. Handling silent data corruption with the sparse grid combination technique. In *Software for Exascale Computing-SPPEXA 2013-2015*. Springer, 2016. pp. 187–208.

[138] Weaver C and Austin TM. A fault tolerant approach to microprocessor design. In *Proceedings of the 2001 International Conference on Dependable Systems and Networks (Formerly: FTCS)*. USA: IEEE Computer Society. ISBN 0769511015.

[139] Iyer RK, Nakka NM, Kalbarczyk ZT et al. Recent advances and new avenues in hardware-level reliability support. *IEEE Micro* 2005; 25(6): 18–29. DOI: 10.1109/MM.2005.119.

[140] von Neumann J. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies* 1956; : 43–98.

[141] Schölzel M. Reduced triple modular redundancy for built-in self-repair in vliw-processors. In *Signal Processing Algorithms, Architectures, Arrangements, and Applications SPA 2007*. pp. 21–26. DOI:10.1109/SPA.2007.5903294.

[142] Austin TM. DIVA: A reliable substrate for deep submicron microarchitecture design. In *Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture, MICRO 32, IEEE Computer Society, Washington, DC, USA*.

[143] Vijaykumar TN, Pomeranz I and Cheng K. Transient-fault recovery using simultaneous multithreading. In *In proceedings of the 29th annual international symposium on computer architecture*. IEEE Computer Society, pp. 87–98.

[144] Benoit A, Cavelan A, Cappello F et al. Identifying the right replication level to detect and correct silent errors at scale. In *FTXS '17: Proceedings of the 2017 Workshop on Fault-Tolerance for HPC at Extreme Scale*. pp. 31–38. DOI: 10.1145/3086157.3086162.

[145] Ainsworth M and Glusa C. Is the multigrid method fault tolerant? the multilevel case. *SIAM Journal on Scientific Computing* 2017; 39(6): C393–C416.

[146] Oh N, Shirvani PP and McCluskey EJ. Error detection by duplicated instructions in super-scalar processors. *IEEE Transactions on Reliability* 2002; 51(1): 63–75. DOI:10.1109/24.994913.

[147] Hoemmen M and Heroux MA. Fault-tolerant iterative methods via selective reliability. *Proceedings of the IEEE/ACM conference on supercomputing (SC'11)* 2011; URL `http://www.researchgate.net/publication/228940928_Fault-Tolerant_Iterative_Methods_via_Selective_Reliability/file/79e41508060305e4ad.pdf`.

[148] Bridges PG, Ferreira KB, Heroux MA et al. Fault-tolerant linear solvers via selective reliability, 2012. `1206.1390`.

[149] Huang KH and Abraham J. Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on Computers* 1984; 100(6): 518–528.

[150] Y Kim, Plank JS and Dongarra JJ. Fault tolerant matrix operations using checksum and reverse computation. In *Proceedings of 6th Symposium on the Frontiers of Massively Parallel Computation (Frontiers '96)*. pp. 70–77. DOI: 10.1109/FMPC.1996.558063.

[151] Du P, Bouteiller A, Bosilca G et al. Algorithm-based fault tolerance for dense matrix factorizations. *ACM SIGPLAN notices* 2012; 47(8): 225–234.

[152] Bosilca G, Delmas R, Dongarra J et al. Algorithm-based fault tolerance applied to high performance computing. *Journal of Parallel and Distributed Computing* 2009; 69(4): 410 – 416.

[153] Chen Z and Dongarra J. Algorithm-based fault tolerance for fail-stop failures. *IEEE Transactions on Parallel and Distributed Systems* 2008; 19(12): 1628–1641.

[154] Jia Y, Bosilca G, Luszczek P et al. Parallel reduction to Hessenberg form with algorithm-based fault tolerance. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC '13, New York, NY, USA: Association for Computing Machinery. DOI:10.1145/2503210.2503249. URL https://doi.org/10.1145/2503210.2503249.

[155] Kabir U and Goswami D. An ABFT scheme based on communication characteristics. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. pp. 515–523. DOI:10.1109/CLUSTER.2016.68.

[156] Cavelan A and Ciorba FM. Algorithm-based fault tolerance for parallel stencil computations. In *IEEE International Conference on Cluster Computing (Cluster 2019)*. Albuquerque.

[157] Moldaschl M, Prikopa KE and Gansterer WN. Fault tolerant communication-optimal 2.5D matrix multiplication. *J Parallel Distributed Comput* 2017; 104: 179–190. DOI:10.1016/j.jpdc.2017.01.022. URL https://doi.org/10.1016/j.jpdc.2017.01.022.

[158] Göddeke D, Altenbernd M and Ribbrock D. Fault-tolerant finite-element multigrid algorithms with hierarchically compressed asynchronous checkpointing. *Parallel Computing* 2015; 49: 117–135. DOI:10.1016/j.parco.2015.07.003.

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

[160] Pachajoa C and Gansterer WN. On the resilience of conjugate gradient and multigrid methods to node failures. In Heras DB, Bougé L, Mencagli G et al. (eds.) *Euro-Par 2017: Parallel Processing Workshops - Euro-Par 2017 International Workshops, Santiago de Compostela, Spain, August 28-29, 2017, Revised Selected Papers*, *Lecture Notes in Computer Science*, volume 10659. Springer, pp. 569–580. DOI:10.1007/978-3-319-75178-8_46. URL https://doi.org/10.1007/978-3-319-75178-8_46.

[161] Agullo E, Giraud L, Guermouche A et al. Numerical recovery strategies for parallel resilient Krylov linear solvers. *Numerical Linear Algebra with Applications* 2016; 23(5): 888–905. DOI:10.1002/nla.2059. URL https://hal.inria.fr/hal-01323192.

[162] Pachajoa C, Levonyak M and Gansterer WN. Extending and evaluating fault-tolerant preconditioned conjugate gradient methods. In *IEEE/ACM 8th Workshop on Fault Tolerance for HPC at eXtreme Scale, FTXS@SC 2018, Dallas, TX, USA, November 16, 2018*. IEEE, pp. 49–58. DOI:10.1109/FTXS.2018.00009. URL https://doi.org/10.1109/FTXS.2018.00009.

[163] Levonyak M, Pacher C and Gansterer WN. 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*. SIAM, pp. 81–92. DOI:10.1137/1.9781611976137.8. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611976137.8. https://epubs.siam.org/doi/pdf/10.1137/1.9781611976137.8.

[164] Pachajoa C, Levonyak M, Gansterer WN et al. How to make the preconditioned conjugate gradient method resilient against multiple node failures. In *Proceedings of the 48th International Conference on Parallel Processing, ICPP 2019, Kyoto, Japan, August 05-08, 2019*. ACM, pp. 67:1–67:10. DOI:10.1145/3337821.3337849. URL https://doi.org/10.1145/3337821.3337849.

[165] Pachajoa C, Pacher C and Gansterer WN. Node-failure-resistant preconditioned conjugate gradient method without replacement nodes. In *2019 IEEE/ACM 9th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS)*. pp. 31–40. DOI:10.1109/FTXS49593.2019.00009.

[166] Langou J, Chen Z, Bosilca G et al. Recovery patterns for iterative methods in a parallel unstable environment. *SIAM J Sci Comput* 2007; 30: 102–116. DOI:10.1137/040620394.

[167] Agullo E, Giraud L, Salas P et al. Interpolation-restart strategies for resilient eigensolvers. *SIAM Journal on Scientific Computing* 2016; 38(5): C560–C583. DOI:10.1137/15M1042115. URL https://hal.inria.fr/hal-01347793.

[168] Jaulmes L, Casas M, ans E Ayguadé MM et al. Exploiting asynchrony from exact forward recovery for detected and uncorrected errors in iterative solvers. In *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. Austin, TX.

[169] Agullo E, Giraud L and Zounon M. On the resilience of parallel sparse hybrid solvers. In *HiPC 2015 - IEEE International Conference on High Performance Computing*. Bangalore, India. URL https://hal.inria.fr/hal-01256316.

[170] Speck R and Ruprecht D. Toward fault-tolerant parallel-in-time integration with PFASST. *Parallel Computing* 2017; 62: 20–37.

[171] Hülsemann F, Kowarschik M, Mohr M et al. Parallel geometric multigrid. In *Numerical Solution of Partial Differential Equations on Parallel Computers*. Springer, 2006. pp. 165–208.

[172] Gmeiner B, Rüde U, Stengel H et al. Towards textbook efficiency for parallel multigrid. *Numerical Mathematics: Theory, Methods and Applications* 2015; 8(1): 22–46.

[173] Rüde U. Error estimators based on stable splittings. In Keyes DE and Xu J (eds.) *Domain Decomposition Methods in Scientific and Engineering Computing: Proceedings of the Seventh International Conference on Domain Decomposition, October 27-30, 1993, the Pennsylvania State University*, *Contemporary Mathematics*, volume 180. American Mathematical Soc., 1994. pp. 111–118.

[174] Huber M, Rüde U and Wohlmuth B. Adaptive control in roll-forward recovery for extreme scale multigrid. *The International Journal of High Performance Computing Applications* 2019; 33(5): 817–837.

[175] Drzisga D, John L, Rüde U et al. On the analysis of block smoothers for saddle point problems. *SIAM Journal on Matrix Analysis and Applications* 2018; 39(2): 932–960. DOI:10.1137/16M1106304. URL `https://doi.org/10.1137/16M1106304`. `https://doi.org/10.1137/16M1106304`.

[176] Schornbaum F and Rude U. Massively parallel algorithms for the lattice Boltzmann method on nonuniform grids. *SIAM Journal on Scientific Computing* 2016; 38(2): C96–C126.

[177] Schornbaum F and Rüde U. Extreme-scale block-structured adaptive mesh refinement. *SIAM Journal on Scientific Computing* 2018; 40(3): C358–C387.

[178] Bauer M, Eibl S, Godenschwager C et al. walberla: A block-structured high-performance framework for multiphysics simulations. *Computers & Mathematics with Applications* 2020; .

[179] Altenbernd M, Dreyer NA, Engwer C et al. Towards local-failure local-recovery in PDE frameworks. In *Proceedings of HPCSE'19*. Springer. Accepted.

[180] Di S and Cappello F. Fast error-bounded lossy HPC data compression with SZ. In *2016 IEEE IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, pp. 730–739.

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

[182] Liang X, Di S, Tao D et al. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. *2018 IEEE International Conference on Big Data (Big Data)* 2018; : 438–447.

[183] Tao D, Di S, Liang X et al. Improving performance of iterative methods by lossy checkpointing. *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing* 2018; DOI:10.1145/3208040.3208050. URL http://dx.doi.org/10.1145/3208040.3208050.

[184] Parra Hinojosa A, Kowitz C, Heene M et al. Towards a fault-tolerant, scalable implementation of GENE. In *Recent Trends in Computational Engineering-CE2014*. Springer, 2015. pp. 47–65.

[185] Stals L. Algorithm-based fault recovery of adaptively refined parallel multigrid grids. *The International Journal of High Performance Computing Applications* 2019; 33(1): 189–211.

[186] Stals L. *Parallel multigrid on unstructured grids using adaptive finite element methods*. PhD Thesis, Department Of Mathematics, The Australian National University, Australia, 1995.

[187] Casas M, Gansterer WN and Wimmer E. Resilient gossip-inspired all-reduce algorithms for high-performance computing: Potential, limitations, and open questions. *IJHPCA* 2019; 33(2). DOI:10.1177/1094342018762531. URL https://doi.org/10.1177/1094342018762531.

[188] Niederbrucker G and Gansterer WN. Robust gossip-based aggregation: A practical point of view. In Sanders P and Zeh N (eds.) *Proceedings of the 15th Meeting on Algorithm Engineering and Experiments, ALENEX 2013, New Orleans, Louisiana, USA, January 7, 2013*. SIAM, pp. 133–147. DOI:10.1137/1.9781611972931.12. URL https://doi.org/10.1137/1.9781611972931.12.

[189] Gansterer WN, Niederbrucker G, Straková H et al. Robust distributed orthogonalization based on randomized aggregation. In Alexandrov VN, Geist A and Dongarra JJ (eds.) *Proceedings of the second workshop on Scalable algorithms for large-scale systems, ScalA@SC 2011, Seattle, WA, USA, November 14, 2011*. ACM, pp. 7–10. DOI:10.1145/2133173.2133177. URL https://doi.org/10.1145/2133173.2133177.

[190] Gansterer WN, Niederbrucker G, Straková H et al. Scalable and fault tolerant orthogonalization based on randomized distributed data aggregation. *J Comput Sci* 2013; 4(6): 480–488. DOI:10.1016/j.jocs.2013.01.006. URL https://doi.org/10.1016/j.jocs.2013.01.006.

[191] Straková H, Niederbrucker G and Gansterer WN. Fault tolerance properties of gossip-based distributed orthogonal iteration methods. In Alexandrov VN,

Lees M, Krzhizhanovskaya VV et al. (eds.) *Proceedings of the International Conference on Computational Science, ICCS 2013, Barcelona, Spain, 5-7 June, 2013*, *Procedia Computer Science*, volume 18. Elsevier, pp. 189–198. DOI:10.1016/j.procs.2013.05.182. URL https://doi.org/10.1016/j.procs.2013.05.182.

[192] Prikopa KE and Gansterer WN. Fault-tolerant least squares solvers for wireless sensor networks based on gossiping. *J Parallel Distributed Comput* 2020; 136: 52–62. DOI:10.1016/j.jpdc.2019.09.006. URL https://doi.org/10.1016/j.jpdc.2019.09.006.

[193] Anzt H, Dongarra J and Quintana-Ortí ES. Tuning stationary iterative solvers for fault resilience. In *Proceedings of the 6th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*. New York, NY, USA: Association for Computing Machinery. ISBN 9781450340113. DOI:10.1145/2832080.2832081. URL https://doi.org/10.1145/2832080.2832081.

[194] Anzt H, Dongarra J and Quintana-Ortí E. Fine-grained bit-flip protection for relaxation methods. *Journal of Computational Science* 2019; 36: 100583. DOI:https://doi.org/10.1016/j.jocs.2016.11.013. URL http://www.sciencedirect.com/science/article/pii/S1877750316303891.

[195] Baudet G. Asynchronous iterative methods for multiprocessors. *Journal of the Association for Computing Machinery* 1978; 25: 226 – 244.

[196] Spitéri P. Parallel asynchronous algorithms for solving boundary value problems. In et al MC (ed.) *Parallel Algorithms*. North-Holland, 1986. pp. 73–84.

[197] Bertsekas D. Distributed asynchronous computation of fixed points. *Math Programming* 1983; 27: 107 – 120.

[198] Bertsekas D and Tsitsiklis J. *Parallel and Distributed Computation, Numerical Methods*. Englewood Cliffs N.J.: Prentice Hall, 1989.

[199] Baz DE, Spitéri P, Miellou J et al. Asynchronous iterative algorithms with flexible communication for non linear network flow problems. *Journal of Parallel and Distributed Computing* 1996; 38: 1–15.

[200] Szyld DB. The mystery of asynchronous iterations convergence when the spectral radius is one. Technical Report 98-102, Department of Mathematics, Temple University, Philadelphia, Pa., 1998. Available at http://www.math.temple.edu/szyld.

[201] Frommer A and Szyld DB. On asynchronous iterations. *Journal of Computational and Applied Mathematics* 2000; 123: 201–216.

[202] Spiteri P. Parallel asynchronous algorithms: A survey. *Advances in Engineering Software* 2020; 149: 1 – 34.

[203] Klawonn A, Kühn MJ and Rheinbach O. Parallel adaptive FETI-DP using lightweight asynchronous dynamic load balancing. *International Journal for Numerical Methods in Engineering* 2020; 121(4): 621–643. DOI:10.1002/nme.6237. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.6237. https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.6237.

[204] Chazan D and Miranker W. Chaotic relaxation. *Linear Algebra and its Applications* 1969; 2(2): 199 – 222. DOI:https://doi.org/10.1016/0024-3795(69)90028-7. URL http://www.sciencedirect.com/science/article/pii/0024379569900287.

[205] Szyld DB. Perspectives on asynchronous computations for fluid flow problems. In Bathe KJ (ed.) *Computational Fluid and Solid Mechanics*. Elsevier, pp. 377–380.

[206] Chow E and Patel A. Fine-grained parallel incomplete LU factorization. *SIAM J Scientific Computing* 2015; 37(2). DOI:10.1137/140968896. URL https://doi.org/10.1137/140968896.

[207] Chow E, Anzt H and Dongarra JJ. Asynchronous iterative algorithm for computing incomplete factorizations on GPUs. In Kunkel JM and Ludwig T (eds.) *High Performance Computing - 30th International Conference, ISC High Performance 2015, Frankfurt, Germany, July 12-16, 2015, Proceedings*, *Lecture Notes in Computer Science*, volume 9137. Springer, pp. 1–16. DOI:10.1007/978-3-319-20119-1_1. URL https://doi.org/10.1007/978-3-319-20119-1_1.

[208] Coleman E, Sosonkina M and Chow E. Fault tolerant variants of the fine-grained parallel incomplete LU factorization. In Polok L, Sosonkina M, Thacker WI et al. (eds.) *Proceedings of the 25th High Performance Computing Symposium, Virginia Beach, VA, USA, April 23 - 26, 2017*. ACM, pp. 15:1–15:12. URL http://dl.acm.org/citation.cfm?id=3108111.

[209] Yamazaki I, Chow E, Bouteiller A et al. Performance of asynchronous optimized Schwarz with one-sided communication. *Parallel Comput* 2019; 86: 66–81. DOI:10.1016/j.parco.2019.05.004. URL https://doi.org/10.1016/j.parco.2019.05.004.

[210] Magoulès F, Szyld DB and Venet C. Asynchronous optimized Schwarz methods with and without overlap. *Numerische Mathematik* 2017; 137: 199–227.

[211] Glusa C, Boman EG, Chow E et al. Scalable asynchronous domain decomposition solvers. Technical Report 19-10-11, Department of Mathematics, Temple University, 2019.

[212] El Haddad M, Garay JC, Magoulès F et al. Synchronous and asynchronous optimized Schwarz methods for one-way subdivision of bounded domains. *Numerical Linear Algebra and Applications* 2020; 27: e2279. 30 pages.

[213] Garay JC, Magoulès F and Szyld DB. Synchronous and asynchronous optimized Schwarz method for Poisson's equation in rectangular domains. Technical Report 17-10-18, Department of Mathematics, Temple University, 2017. Revised April 2018.

[214] Rüde U. Fully adaptive multigrid methods. *SIAM Journal on Numerical Analysis* 1993; 30(1): 230–248.

[215] Southwell RV. *Relaxation methods in engineering science, a treatise on approximate computation.* Oxford, Oxford Univ. Pr., 1946. 1.ed., reprint.

[216] Rüde U. *Mathematical and computational techniques for multilevel adaptive methods*. SIAM, 1993.

[217] Szyld DB and Xu JJ. Convergence of some asynchronous nonlinear multisplitting methods. *Numerical Algorithms* 2000; 25: 347–361.

[218] Kollias G, Gallopoulos E and Szyld DB. Asynchronous iterative computations with Web information retrieval structures: The PageRank case. In Joubert G, Nagel W, Peters F et al. (eds.) *Parallel Computing: Current and Future Issues of High-End Computing (Proceedings of the International Conference Parco05)*, *NIC Series*, volume 33. Jülich, Germany: John von Neumann-Institut für Computing (NIC), pp. 309–316.

[219] Wolfson-Pou J and Chow E. Asynchronous multigrid methods. *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* 2019; : 101–110.

[220] Sao P and Vuduc R. Self-stabilizing iterative solvers. *Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems - ScalA '13* 2013; : 1–8DOI:10.1145/2530268.2530272. URL `http://dl.acm.org/citation.cfm?doid=2530268.2530272`.

[221] Dijkstra EW. Self-stabilizing systems in spite of distributed control. *Communications of the ACM* 1974; 17(11): 643–644.

[222] Zoutendijk G. Nonlinear programming, computational methods. *Integer and nonlinear programming* 1970; : 37–86.

[223] Elliott J. *Resilient Iterative Linear Solvers Running Through Errors.* PhD Thesis, North Carolina State University, 2015.

[224] Elliott J, Hoemmen M and Mueller F. Evaluating the impact of SDC on the GMRES iterative solver. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. pp. 1193–1202. DOI:10.1109/IPDPS.2014.123.

[225] Griebel M and Oswald P. Stochastic subspace correction methods and fault tolerance. *Mathematics of Computation* 2020; 89(321): 279–312.

[226] Glusa C, Ramanan P, Boman EG et al. Asynchronous one-level and two-level domain decomposition solvers. *CoRR* 2018; abs/1808.08172. URL http://arxiv.org/abs/1808.08172. 1808.08172.

[227] Rizzi F, Morris K, Sargsyan K et al. ULFM-MPI implementation of a resilient task-based partial differential equations preconditioner. In *Proceedings of the ACM Workshop on Fault-Tolerance for HPC at Extreme Scale*. New York, NY, USA: Association for Computing Machinery. ISBN 9781450343497. DOI:10.1145/2909428.2909429. URL https://doi.org/10.1145/2909428.2909429.

[228] Rizzi F, Morris K, Sargsyan K et al. Exploring the interplay of resilience and energy consumption for a task-based partial differential equations preconditioner. *Parallel Computing* 2018; 73: 16 – 27. DOI:https://doi.org/10.1016/j.parco.2017.05.005. URL http://www.sciencedirect.com/science/article/pii/S0167819117300753. Parallel Programming for Resilience and Energy Efficiency.

[229] Morris K, Rizzi F, Cook B et al. Performance scaling variability and energy analysis for a resilient ULFM-based PDE solver. In *2016 7th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*. pp. 41–48. DOI:10.1109/ScalA.2016.010.

[230] Chen S, Bronevetsky G, Casas-Guix M et al. Comprehensive algorithmic resilience for numeric applications. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2013.

[231] Poulos A, Wallace D, Robey R et al. Improving application resilience by extending error correction with contextual information. In *2018 IEEE/ACM 8th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS)*. pp. 19–28. DOI:10.1109/FTXS.2018.00006.

[232] Diaz JM, Pophale S, Friedline K et al. Evaluating support for openmp offload features. In *Proceedings of the 47th International Conference on Parallel Processing Companion*. pp. 1–10.

[233] Maghraoui K, Desell T, Szymanski B et al. Dynamic malleability in iterative MPI applications. In *In: 7th International Symposium on Cluster Computing and the Grid*. pp. 591–598. DOI:10.1109/CCGRID.2007.45.

[234] Buchwald S, Mohr M and Zwinkau A. Malleable invasive applications. *CEUR Workshop Proceedings* 2015; 1337: 123–126.

[235] Farhat C and Roux FX. A method of finite element tearing and interconnecting and its parallel solution algorithm. *International Journal for Numerical Methods in Engineering* 1991; 32(6): 1205–1227.

[236] Alefeld G and Herzberger J. *Introduction to interval analysis*. Academic Press, 1983.

[237] Kulisch U. *Advanced Arithmetic for the Digital Computer*. Springer, Wien, 2002.

[238] Vignes J. Discrete Stochastic Arithmetic for validating results of numerical software. *Numerical Algorithms* 2004; 37(1–4): 377–390.

[239] Parker DS, Pierce B and Eggert PR. Monte Carlo arithmetic: a framework for the statistical analysis of roundoff error. *IEEE Computation in Science and Engineering* 2001; .

[240] Frechtling M and Leong PHW. MCALIB: measuring sensitivity to rounding error with monte carlo programming. *ACM TOPLAS* 2015; 37(2): 1–25.

[241] Denis C, de Oliveira Castro P and Petit E. Verificarlo: checking floating point accuracy through Monte Carlo arithmetic. In *ARITH'23, Silicon Valley, USA*.

[242] Févotte F and Lathuilière B. Debugging and optimization of HPC programs in mixed precision with the verrou tool. In *Computational Reproducibility at Exascale Workshop (CRE2018), in conjunction with the International Conference on High Performance Computing, Networking, Storage and Analysis (SC18), Dallas, USA*.

[243] Eberhart P, Brajard J, Fortin P et al. High performance numerical validation using stochastic arithmetic. *Reliable Computing* 2015; 21: 35–52.

[244] Bank RE and Smith RK. A posteriori error estimates based on hierarchical bases. *SIAM Journal on Numerical Analysis* 1993; 30(4): 921–935.

[245] Quarteroni A and Valli A. *Numerical approximation of partial differential equations*, volume 23. Springer Science & Business Media, 2008.

[246] Ainsworth M and Oden JT. *A posteriori error estimation in finite element analysis*, volume 37. John Wiley & Sons, 2011.

[247] Karniadakis G and Sherwin S. *Spectral/hp element methods for computational fluid dynamics*. Oxford University Press, 2013.

[248] Lambert J. *Numerical Methods for Ordinary Differential Systems*. Chirchester, England: Wiley, 1991.

[249] Hairer E, Nørsett S and Wanner G. *Solving Ordinary Differential Equations I: Nonstiff Problems*. 3rd corr. ed. Berlin Heidelberg: Springer-Verlag, 2008.

[250] Gear C and Wells D. Multirate linear multistep methods. *BIT* 1984; 24: 484–502.

[251] Savcenco V, Hundsdorfer W and Verwer J. A multirate time stepping strategy for stiff ordinary differential equations. *BIT* 2007; 47: 137–155.

[252] Seny B, Lambrechts J, Toulorge T et al. An efficient parallel implementation of explicit multirate Runge–Kutta schemes for discontinuous Galerkin computations. *Journal of Computational Physics* 2014; 256: 135–160.

[253] Fok P. A linearly fourth order multirate Runge–Kutta method with error control. *Journal of Scientific Computing* 2015; : 1–19.

[254] Sandu A. A class of multirate infinitesimal gark methods. *SIAM Journal on Numerical Analysis* 2019; 57(5): 2300–2327.

[255] Carciopolo LD, Bonaventura L, Scotti A et al. A conservative implicit multirate method for hyperbolic problems. *Computational Geosciences* 2019; 23: 647–664.

[256] Bonaventura L, Casella F, Carciopolo LD et al. A self adjusting multirate algorithm for robust time discretization of partial differential equations. *Computers and Mathematics with Applications* 2020; 79: 2086–2098.

[257] Bangerth W, Burstedde C, Heister T et al. Algorithms and data structures for massively parallel generic adaptive finite element codes. *ACM Transactions on Mathematical Software (TOMS)* 2012; 38(2): 1–28.

[258] Bangerth W and Rannacher R. *Adaptive finite element methods for differential equations*. Birkhäuser, 2013.

[259] Piggott M, Farrell P, Wilson C et al. Anisotropic mesh adaptivity for multi-scale ocean modelling. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 2009; 367(1907): 4591–4611.

[260] Berger MJ, George DL, LeVeque RJ et al. The GeoClaw software for depth-averaged flows with adaptive refinement. *Advances in Water Resources* 2011; 34(9): 1195–1206.

[261] LeVeque RJ, George DL and Berger MJ. Tsunami modelling with adaptively refined finite volume methods. *Acta Numerica* 2011; 20: 211–289.

[262] Müller A, Behrens J, Giraldo FX et al. Comparison between adaptive and uniform discontinuous Galerkin simulations in dry 2D bubble experiments. *Journal of Computational Physics* 2013; 235: 371–393.

[263] Tumolo G and Bonaventura L. A semi-implicit, semi-Lagrangian discontinuous Galerkin framework for adaptive numerical weather prediction. *Quarterly Journal of the Royal Meteorological Society* 2015; 141(692): 2582–2601.

[264] Bauer S, Mohr M, Rüde U et al. A two-scale approach for efficient on-the-fly operator assembly in massively parallel high performance multigrid codes. *Applied Numerical Mathematics* 2017; 122: 14–38.

[265] Bauer S, Drzisga D, Mohr M et al. A stencil scaling approach for accelerating matrix-free finite element implementations. *SIAM Journal on Scientific Computing* 2018; 40(6): C748–C778.

[266] Lengauer C, Apel S, Bolten M et al. ExaStencils – Advanced Multigrid Solver Generation. In Bungartz HJ, Reiz S, Neumann P et al. (eds.) *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.

[267] Bauer S, Huber M, Ghelichkhan S et al. Large-scale simulation of mantle convection based on a new matrix-free approach. *Journal of Computational Science* 2019; 31: 60–76.

[268] Guan Q, Debardeleben N, Blanchard S et al. F-SEFI: A fine-grained soft error fault injection tool for profiling application vulnerability. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. pp. 1245–1254. DOI:10.1109/IPDPS.2014.128.

[269] Jin A, Jiang J, Hu J et al. A pin-based dynamic software fault injection system. In *2008 The 9th International Conference for Young Computer Scientists*. pp. 2160–2167. DOI:10.1109/ICYCS.2008.329.

[270] Hari SKS, Tsai T, Stephenson M et al. 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*. IEEE Computer Society, pp. 249–258. DOI:10.1109/ISPASS.2017.7975296. URL https://doi.org/10.1109/ISPASS.2017.7975296.

[271] Calhoun J, Olson L and Snir M. Flipit: An llvm based fault injector for hpc. In *Revised Selected Papers, Part I, of the Euro-Par 2014 International Workshops on Parallel Processing - Volume 8805*. Berlin, Heidelberg: Springer-Verlag. ISBN 9783319143248, p. 547–558. DOI:10.1007/978-3-319-14325-5_47. URL https://doi.org/10.1007/978-3-319-14325-5_47.

[272] Georgakoudis G, Laguna I, Nikolopoulos DS et al. 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: Association for Computing Machinery. ISBN 9781450351140. DOI:10.1145/3126908.3126972. URL https://doi.org/10.1145/3126908.3126972.

[273] Menon H and Mohror K. DisCVar: Discovering critical variables using algorithmic differentiation for transient faults. *SIGPLAN Not* 2018; 53(1).

DOI:10.1145/3200691.3178502. URL https://doi.org/10.1145/3200691.3178502.

[274] Reis GA, Chang J, Vachharajani N et al. Swift: software implemented fault tolerance. In *International Symposium on Code Generation and Optimization*. pp. 243–254.

[275] Lin S and Chen P. A simd-based software fault tolerance for arm processors. In *2017 International Conference on Applied System Innovation (ICASI)*. pp. 910–913.

[276] Lidman J, Quinlan DJ, Liao C et al. 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)*. pp. 1–6. DOI:10.1109/DSNW.2012.6264672.

[277] Rodríguez G, Martín MJ, González P et al. Cppc: A compiler-assisted tool for portable checkpointing of message-passing applications. *Concurr Comput: Pract Exper* 2010; 22(6): 749–766.

[278] Hukerikar S, Teranishi K, Diniz PC et al. An evaluation of lazy fault detection based on adaptive redundant multithreading. In *2014 IEEE High Performance Extreme Computing Conference (HPEC)*. pp. 1–6.