Fault-tolerant finite-element multigrid algorithms with hierarchically compressed asynchronous checkpointing

Dominik Göddeke[a,*], Mirco Altenbernd[a], Dirk Ribbrock[b]

[a] *Institute for Applied Analysis and Numerical Simulation, University of Stuttgart, Pfaffenwaldring 57, 70569 Stuttgart, Germany*
[b] *Fakultät für Mathematik (LS3), TU Dortmund, Vogelpothsweg 87, 44227 Dortmund, Germany*

Abstract

We examine novel fault tolerance schemes for data loss in multigrid solvers which essentially combine ideas of checkpoint-restart with algorithm-based fault tolerance. To improve efficiency compared to conventional global checkpointing, we exploit the inherent data compression of the multigrid hierarchy, and relax the synchronicity requirement through a local failure local recovery approach. We experimentally identify the root cause of convergence degradation in the presence of data loss using smoothness considerations. Our resulting schemes form a family of techniques, that can be tailored to the expected error probability of (future) large-scale machines. A performance model gives further insight into the benefits and applicability of our techniques.

Keywords: fault tolerance, resilience, multigrid, checkpoint-restart, robust iterative solvers, high-performance computing

1. Introduction

1.1. Parallel multigrid methods

Multigrid methods (MG, [1, 2]) are one of the most favourable approaches for the iterative solution of elliptic or quasi-elliptic discretised PDEs (partial differential equations). In combination with implicit time-stepping for instationary problems, and Newton-type methods for treating nonlinearities, hierarchical multigrid techniques are the only class of solvers with the potential to deliver asymptotically optimal and hence algorithmically scalable convergence. Multigrid approaches can solve the resulting sparse, but typically huge linear systems of equations in a number of iterations that is independent of the mesh width – and hence the problem size – of the underlying discretisation. Application domains in which the solution of such linear systems dominates the simulation time include but are certainly not limited to fluid dynamics (e.g., Pressure-Poisson problems arising in solution schemes for the Navier-Stokes equations), elasticity (e.g., linear and nonlinear Lamé equations), and electrostatics and capacitance tomography (e.g., potential calculations using generalised Poisson equations).

[*] Corresponding author
Email address: dominik.goeddeke@mathematik.uni-stuttgart.de (Dominik Göddeke)

When combined with the finite element method (FEM) for the discretisation, the various components of multigrid methods can be tailored beneficially to the problem at hand, e.g., to satisfy the smoothing property even for strongly anisotropic meshes or operators, or to provably meet a best-approximation criterion for the coarse-grid correction.

For large-scale problems, FE-MG methods are typically realised in a data-parallel fashion: The computational domain is partitioned into patches (subdomains), yielding coarse-grained parallelism. With the exception of certain smoothers and the coarse grid solver, all operations like residual calculations or transfers of relevant quantities between coarse and fine grids, are entirely local, or involve communication of patch surface data with neighbouring patches only. For the coarse grid problem – and more generally for the entire coarsening process in the preprocessing stage of algebraic multigrid methods – techniques like replicated solves, or leaving an increasing amount of processors idle, are often used to avoid quasi-serialisation and unfavourable communication-to-computation ratios [3]. These techniques are however not applicable to some of the most powerful and robust smoothers, e.g., implicit approaches based on incomplete factorisations. They entail their favourable numerical properties from their recursive nature or strong degree of global coupling, resulting in serial data dependencies, computationally inefficient execution, and poor scalability. To alleviate this issue, most parallel multigrid methods replace the global smoother with an additive or multiplicative Schwarz approach [4], which limits strong coupling and irregular computations to within patches. This is justified, since the bulk of computations are typically performed in the smoother stage. However, the resulting block-Jacobi or block-Gauß-Seidel characteristics may lead to convergence degradation for hard problems, e.g., in the presence of strong anisotropies in the mesh and/or the operators.

In our ScaRC framework (scalable recursive clustering) [5], a generalisation of geometric multigrid and multilevel domain decomposition, we try to balance these contradicting goals as much as possible, and apply a number of additional techniques to improve efficiency: We cover the domain with an unstructured coarse mesh, and refine each patch in a structured way. Several patches are combined into an MPI rank, and the structure of the patches is exploited in the design of specifically tailored multigrid components, incorporating PDE- and problem knowledge as well as architectural considerations, e.g., for GPUs or to increase locality [6]. The block-smoother can recursively cluster several neighbouring patches into one 'virtual' patch, and by applying a few cycles of a (semi-) local multigrid method recursively as a smoother of the global multilevel scheme, efficiency and especially numerical scalability are significantly improved. The globally-unstructured locally-structured mesh *and* solver structure thus constitutes a good compromise between parallel and numerical scalability, fast convergence, and hardware exploitation. The ScaRC approach is also highly beneficial for the fault-tolerant schemes we present in this paper.

1.2. Hardware development and reliable computing

Researchers are becoming increasingly aware that future computing systems may become very unreliable, at least compared to current standards. In fact, this issue has been identified as one of the main obstacles on the road to exascale computing [7–9]. On the one hand, the vast number of components in present and future computing systems implies a strong increase in fault probability. On the other hand, the continuing shrinking of photolithography resolution leads to greater vulnerability to leakage current and other interferences like radiation, resulting in the flipping of bits that may (silently) corrupt calculation outcomes and (intermediate and solution) data in various memories.

In view of the power wall problem and the implied tight power envelopes, hardware developers are even actively debating a trade-off between the demand for higher performance through more processing units, and a reduction of hardware-based error correction mechanisms like ECC (error-correcting-codes) protection of memory and data paths [10], which increases vulnerability and thus fault probability.

It can be expected that these issues will not be limited to future (exascale) leadership-class machines with up to billion-way parallelism and beyond: Similarly to the era of homogeneous clusters (and the 'flat MPI' programming model) that rapidly approaches its end, systems at any reasonable scale might be based on essentially the same technology, and suffering from a similar degree of unreliability.

Following established taxonomy by Elliott et al. and Snir et al. [11, 12], throughout this paper we distinguish between faults, i.e., the abnormal operation of some component, and failures, i.e., the (resulting) computation of a wrong answer or none at all; and also between hard (program termination) and soft faults. Examples range from the flipping of a bit in some circuit (a rather small-scale component) up to the loss of an entire cluster node.

In summary, unreliability is likely to become the norm rather than the exception: On future systems, the mean time between failure (MTBF) might eventually decrease to a point where any simulation run will be affected by some kind of irregular behaviour. Consequently, a certain degree of fault-tolerance will be an obligatory ingredient of any simulation code.

1.3. Paper contribution

In our work, we pursue a 'bottom-up' approach towards fault-tolerant solvers: We argue that not only the software in general, but rather the numerical schemes themselves must ensure reliability and robustness. Especially in the solution of PDE problems, more efficient resilience mechanisms can be constructed by explicitly incorporating properties of the PDE and the solution mechanism at hand to detect and mitigate or correct faults. Classical checkpoint-restart techniques should be considered the 'ultima ratio', and only be used if the numerical scheme alone is not capable of preventing a fault becoming a failure.

We are certainly not the first to argue along these lines, cf. for instance several recent reports [7–9]. In this paper, we describe and experimentally analyse novel techniques towards this goal, focusing mostly on multigrid algorithms and their hierarchical properties. Despite their importance, multigrid methods have received fewer attention in contrast to Krylov subspace schemes [9]. In addition, we believe that the novel techniques we develop can be extended to other hierarchical solution methods, such as multiscale modelling, upscaling etc.

In contrast to the recent surge of techniques on silent data corruption in sparse linear solvers, we more generally investigate the case that in the course of the iterative solve, some portions of the global solution are lost, and this loss needs to be recovered from.

After presenting and discussing some necessary background and related work in Section 2, we specify our evaluation procedure and our fault injection model in Sections 3.1 and 3.2. In particular, we describe how we can model and analyse the large-scale situation numerically based on appropriately scaled-down experiments, using essential properties of multigrid. We then make two main contributions in this paper, all derived from numerical experiments rather than rigorous numerical analysis:

For a representative set of model problems, we analyse the impact of data loss in the multigrid algorithm in the remainder of Section 3. Here, we identify aspects of convergence degradation for the prototypical set of PDEs we examine, and highlight that multigrid is indeed robust in case of infrequent faults: We demonstrate convergence even when substantial portions of the current iterate are lost. Using the terminology of Sao and Vuduc [13], these results indicate that geometric multigrid schemes are inherently 'self-stabilising' with respect to transient faults for elliptic (model) problems.

We then devise a novel fault-tolerant scheme for multigrid in Section 4, by realising that the grid and function space hierarchies can be interpreted as a 'lossy compression' which however still fulfils a best-approximation property. Our proposed scheme incorporates and extends ideas of full multigrid [14] into a combined 'local failure local recovery' [15] and 'algorithm-based fault-tolerance' [16] approach. While still relying on the existence of global, albeit potentially asynchronous and outdated checkpoints, our approach improves efficiency, as these checkpoints can be exponentially smaller in size, and recovery proceeds purely local. Numerical experiments demonstrate the feasibility and the advantages of our approach.

In Section 5 we discuss how the promising numerical behaviour of our techniques can be transferred to practical implementations. Here, we exemplarily limit ourselves to the case that 'data loss' corresponds to the loss of a compute node, which without loss of generality corresponds to the loss of an MPI rank. To assess the overhead of our approach and compare it with more classical, problem-oblivious fault-tolerance techniques, we construct an analytical model in Section 5.2 and examine various tradeoffs.

We conclude with an outlook on ongoing and future work in Section 6.

2. Background and related work

The transition towards exascale (extreme scale) computing has resulted in a host of research efforts, spanning hardware and circuits, systems, middleware, implementation, (numerical) methods and algorithms, and applications. Recently, a number of excellent surveys have been published [7–9, 12]. There is a strong consensus that future machines will be much more unreliable than current ones, and thus fault-tolerance has been identified as one of the main research avenues.

Traditionally, fault-tolerance has been achieved with global checkpoint-restart (CPR) schemes that incorporate globally consistent checkpoints of essential state. However, these methods are known not to scale well [7–9], which may eventually render them prohibitively expensive. Consequently, the so-called local failure local recovery paradigm seems more appropriate [15], and is subject to active research. Our proposed scheme falls into this category. Extensions like checkpointing the full state to the memory of neighbouring nodes are also not optimal, given the expected trend of decreasing memory per core [9].

As an alternative, redundancy combined with majority voting (e.g., triple modular redundancy, TMR) has received increasing interest: Elliott et al. and Fiala et al. argue that for sufficiently large fault probability, TMR can potentially be more energy- and runtime-efficient than traditional synchronous CPR despite the increased resource demands [17, 18]. In view of the power wall problem, redundancy does not seem to be the most practicable approach however, because of the at least tripled resource requirements. As a remedy, partial redundancy combined with classical checkpointing is investigated, see for instance Elliott et al. [17]. Ni et al. and Fiala et al. have implemented these techniques on top of MPI to facilitate automatic fault-tolerance [18, 19].

To alleviate the drawbacks of the global synchronicity and consistency of checkpoints, so-called send- or channel-deterministic algorithms allow asynchronous checkpointing combined with message logging, resulting in a scalable CPR scheme [20] that also allows local recovery. Many PDE solvers fall into this category. Similar research efforts are underway, for instance uncoordinated checkpoint protocols [21, 22]. Multi-level checkpointing combined with efficient on-the-fly data compression is yet another attempt to alleviate the scalability issues of classical CPR [23].

Another avenue can be loosely classified as algorithm-based fault-tolerance (ABFT). This term dates back to early work by Huang and Abraham [16], who employ checksums to detect and eventually correct failures directly in dense matrix-vector multiplications, quite similar to software-based error-correcting-codes (ECC). See Bosilca et al. [24] for an updated study at scale on contemporary machines.

Given this wide variety of resilience techniques, we argue that there is no 'one-size-fits-all' 'black-box' solution to the resilience challenge, at least as soon as runtime efficiency and robustness are the dominant (relevant) criteria. This is especially true for PDE problems. Many efforts have been undertaken to improve the resilience of linear solvers for sparse problems. The majority of this work however focuses on Krylov subspace solvers like GMRES, due to their black-box character [11, 13, 15, 25–29]. Also, most papers address ABFT techniques for so-called silent data corruption, i.e., undetected and uncorrected bitflips. From a numerics point of view, most of these approaches are based on the theory of inner-outer iterations and self-preconditioning combined with flexible formulations of the underlying solvers. The selective reliability mechanism by Hoemmen and Heroux [30] often is the underlying machine model in this context. Multigrid methods have received much less attention. Mishra and Banerjee [31] discuss error detection, and Jimack and Walkley [32] present a good overview on the theoretical background of resilient and in particular asynchronous multigrid solvers. Huber et al. [33] examine an approach of locally repairing lost patches in extreme-scale multigrid solvers, which is quite similar to ours. Their research was undertaken concurrently to ours, and confirms our results: We may rely on the h-independent convergence of multigrid for a more detailed analysis, and can expect quantitatively similar behaviour at scale. Exploiting numerical invariants like orthogonality, positivity or monotonicity to construct fault detection mechanisms is another active field of research.

Realistic numbers on the frequency of hard faults are reported in a number of studies: For instance, Jaguar (Oak Ridge National Labs, TOP500 # 1 in November 2009) exhibits a mean time between failure of 52 hours [18]. Schroeder et al. and Sridharan and Liberty [34, 35] examine fault rates for individual components of the heterogeneous Google server farms. For soft faults, reliable numbers are not generally available. It is however well-known that silicon ageing increases the rate of ECC corrections after an initial fault at the same bank [18]. The ECC mechanism is probably the most well-known example of fault detection and correction implemented in hardware. However, ECC does not protect all data, memory paths, and certainly not the ALUs, so that a bitflip during an arithmetic operation may corrupt the result, see Elliott et al. [36] for an instructive example. Due to the von Neumann architectural paradigm, program code and data are both subject to faults and data corruption. But, since the size (in Bytes) of the executable code is almost always only a tiny fraction of the program specific data, we may with little practical impact concentrate on faults in the program specific data and assume the program code to be error-free. In analogy of the proposed 'selective reliability' mechanism by Hoemmen and Heroux [30] and Bridges et al. [25], we may assume that program code and control flow instructions are stored in and executed by a more reliable, albeit slower and more expensive part of the machine.

Finally, Elliott et al. [11] and Li et al. [37] argue that spontaneous bitflips in DRAM are almost always corrected by ECC if available, and that the uncorrected case is surprisingly rare. Li et al. in particular state that over-protection must be avoided [37]. This last work in particular bridges the gap between hardware-based and algorithm-based fault-tolerance.

3. Impact of data loss in multigrid solvers

3.1. Test problems

We consider the Poisson problem as the classical prototype of elliptic PDEs. As outlined in the introduction, this fundamental model problem is at the core of many application domains and numerical schemes. For an open, bounded domain $\Omega \subset \mathbb{R}^2$ with boundary $\Gamma := \partial\Omega$, we want to solve the Dirichlet boundary value problem

$$\begin{aligned} -\Delta u &= f \qquad \text{in } \Omega \\ u &= g \qquad \text{on } \Gamma \end{aligned} \tag{1}$$

for suitable $u, f : \Omega \setminus \Gamma \to \mathbb{R}$ and $g : \Gamma \to \mathbb{R}$. As in this paper we are primarily interested in the numerical behaviour, we may restrict ourselves to the unit square, $\Omega = [0,1]^2$. We start with a coarse grid of 16 quadrilateral mesh cells, and use seven successive uniform refinement steps to define the mesh hierarchy. On each of these refinement levels, the PDE is discretised with conforming biquadratic Q_2 finite elements, resulting in a fine-grid problem of 1 050 625 unknowns, and 263 169, 525 312, 262 144 vertices, edges and cells, respectively. The unknowns are enumerated in a recursive two-level fashion. We prescribe

$$u_{\text{exact}}(x,y) := \sin(n\pi x)\sin(k\pi y)$$

as the expected solution on $\bar{\Omega}$, and accordingly define the right hand side f of (1) as the analytical evaluation of $-\Delta u_{\text{exact}}$ at the quadrature points, multiplied by the test function and integrated over the elements. In order to quantify errors we use the L^2 norm, i.e., the finite element integration of the difference functional between computed and exact analytical solution. Residuals are measured in the Euclidean 2-norm, scaled so that the initial norm before the start of the solver is exactly one.

Since the core concept of multigrid is the efficient elimination of error frequencies, we consider the *smooth case* $n = 1$ and $k = 3$, and the *oscillatory case* $n = 10$ and $k = 30$. These settings are representative of those encountered in 'real problems', and this approach enables us to argue about the root cause of the convergence behaviour we experimentally observe in the presence of injected faults. The fine-grid resolution is chosen so that all frequencies comprising the exact solution can be accurately resolved, i.e., the discretisation error does not dominate the error due to data loss, which we are primarily interested in. Figure 1 depicts the fine-grid solution of both test problems.

To facilitate wide applicability of our experiments, we employ a standard multigrid solver configuration. We do not expect our results to be fundamentally different for other configurations, as long as our error injection model (see Section 3.2) is applicable. The solver executes V-cycles until the relative residual norm has been reduced by nine digits. We employ two pre- and postsmoothing steps, respectively, with a Jacobi smoother that is damped by $\omega = 0.8$. Restriction and prolongation are realised by biquadratic finite element interpolation, matching the ansatz space, to ensure the best-approximation property. All coarse grid problems are solved to machine accuracy to avoid side effects, using BiCGStab. In all tests, the initial guess is zero.

3.2. Fault injection model and emulation of large-scale computations

We emulate the fully parallel case in our numerical experiments with a serial implementation that scales down all relevant parameters. This is justified, because (parallel) multigrid solvers exhibit convergence rates independent of the mesh and patch width (h and H independence, [4]). This holds at least for our (currently) uniform setting and for Schwarz smoothers, see the sketch of our FEAST/ScaRC solvers in Section 1.1 and the references therein. Thus, the amount of lost data at scale (e.g., one entire compute node) can be realised in our serial prototype implementation by faulting a small connected patch, of the same relative size. In particular, the convergence behaviour is numerically identical.

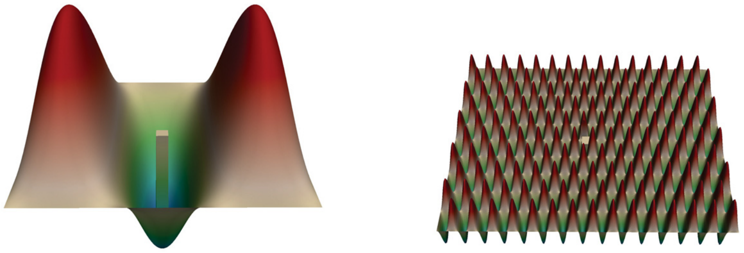

Figure 1: Illustration of our fault injection model, exemplarily for zeroing out some components in the converged fine grid solution of the smooth (left) and oscillatory (right) test problems. In this figure, the patch location and size, and the scaling of the y-axis, have been chosen to improve clarity.

Following standard practice in the growing field of numerical resilience research, we assume that 'data' such as the discretised operators (on all refinement levels) and the right hand side (on the fine grid) are never lost. Since these data are constant during a linear solve, conventional checkpoint-recovery techniques can be used with little overhead: There is no need to periodically checkpoint matrices and the right hand side, and the recovery can be performed fully locally.

In the multigrid setting, auxiliary arrays are only needed to store restricted residuals and prolongated corrections, mostly on coarser levels. We assume that the coarsest-grid problem in any given cycle is solved exactly and without faults. Thus, we can exploit the ellipticity and argue that any small-area loss in any of the auxiliary arrays is propagated to a (slightly) larger data loss in the solution once we again reach the fine grid in the course of single multigrid cycle. The actual size depends on the hierarchy level at which the fault occurs. Consequently, we consider only the case of fault injection into the fine-grid solution, but vary the amount of affected degrees of freedom.

To this end, we randomly choose a vertex of the fine grid, and consider a small layer of elements around this vertex. For all degrees of freedom in this rectangular patch, we invalidate all components of the fine-grid solution. Figure 1 illustrates such an injection into a small patch of the converged fine grid solution around the centre of the domain, exemplarily for zeroing the lost components.

In practice, this procedure obviously requires knowledge about which patch is lost. We return to this topic in Section 5.

3.3. Experimental convergence analysis of single faults

To analyse convergence in the presence of faults, we configure the multigrid solver as described above, and emulate a fault by zeroing 169 (i.e., 0.02 %) degrees of freedom of the fine grid iterate at the end of selected multigrid cycles. For these experiments, we select the centre of the domain as the source of the fault, which constitutes the 'worst case', cf. Figure 1. In our experiments, we found that the smoothness of the test problem has the largest influence, and to a much lesser extent, the size of the fault injection (0.02 to 10 % of the domain) and the type and location of the fault injection (zeros, pseudo-random values in the interval $[-\|u\|_{\max}, \|u\|_{\max}]$, sign flips, and bitflips excluding the high-order exponent bits). Due to space constraints, we only vary the smoothness of the problem, and keep the fault location and size fixed in this experiment.

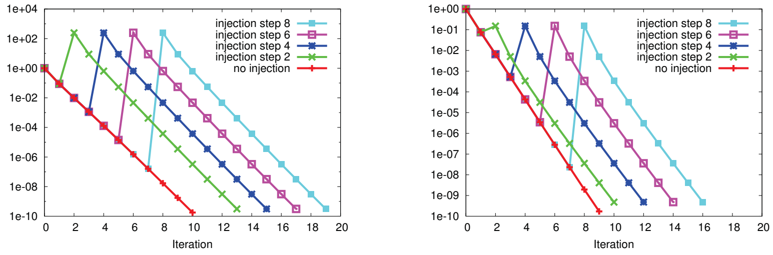

Figure 2: Multigrid convergence in the scaled residual norm (y-axis) for the smooth (left) and oscillatory (right) test problem. Faults are injected at the end of different cycles. We first compute the residual and its norm for convergence control, then inject the fault, and then compute the residual norm again. Only the latter norm is depicted in the plots. For example, the green ('x glyphs') curve corresponds to a fault injection after the second cycle.

Figure 2 depicts the convergence of the multigrid solver, for the smooth and the oscillatory test problem, respectively. Each plot corresponds to injecting the described fault into the current fine-grid iterate after different cycles. The green ('x glyphs', labelled as 'injection step 2') plot for instance corresponds to the case that initially, the solver is permitted to iterate through two cycles in a fault-free fashion. Then, a residual norm is computed for convergence control, followed by the fault injection, and another residual norm computation. Only the latter residual norm is shown in the plots, and hence, the corresponding fault-free residual norm after the cycle can best be derived from the completely fault-free (red, 'plus glyphs') plot.

We first note that all solvers iterate to convergence, and for these experiments, the number of multigrid cycles slightly less than doubles in the worst case. The later we inject a fault, the longer it takes to reach convergence. Using the terminology of Sao and Vuduc [13], these results indicate that geometric multigrid schemes are inherently 'self-stabilising' for elliptic (model) problems with respect to transient, single soft faults. This is an advantage of multigrid methods in contrast to general Krylov subspace schemes such as the conjugate gradient method, where faults in the search directions, and hence the iteratively constructed basis, may lead to divergence, or convergence to the wrong solution. See below for a more theoretical argument.

However, we observe strong jumps in the residual norms, which ultimately cause delayed convergence. Note that these jumps are in our taxonomy the failures caused by the fault. Also, it is remarkable that the jumps all reach the same plateau, independent of the cycle after which the fault is injected. This behaviour is qualitatively identical for both the smooth and the oscillatory test problems, the differences are only quantitative. For the smooth case, the residual norm reaches values that are several orders of magnitude larger than the initial norm.

We briefly comment on some exemplary quantitative differences we observed in the tests not shown: For the smooth problem, the difference compared to the presented case, in the residual norm, is at most one order of magnitude when injecting random values or introducing bitflips. If we increase the patch size up to 10 % of the domain and use bounded random values instead of zeros, the difference increases to at most two orders of magnitude. For the oscillatory problem, we obtain analogous values, with the exception that non-surprisingly, the differences between injecting zeros and random values are slightly more pronounced.

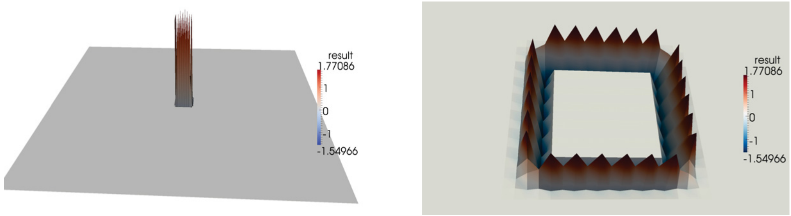

Figure 3: Visualisation of the residual immediately after injecting a fault into the almost converged smooth solution (left), and a zoom on the vicinity of the faulty patch (right).

Theoretical convergence considerations. Convergence in the case of single faults can be shown rigorously. However, we omit a formal proof in this paper, and only sketch its main ideas: First, multigrid is nothing more than a sophisticated defect correction procedure, i.e., a fixed point iteration. In fact, Hackbusch's qualitative multigrid convergence proof [1] is based on contraction arguments for the recursively defined iteration operator (propagation operator). The main argument is that as long as the contraction property holds for a given iteration operator, then convergence of the corresponding iteration procedure is guaranteed for *any* initial guess u_0. This is basically Banach's fixed point theorem, see for instance [38, Section 4.2]. In particular, this means that in the event of a fault, the new initial guess $\tilde{u}_0$ is simply the last iterate u_k with some faulty entries. Since we assume that all matrices and grid transfer operators are not affected, the contraction property is not affected by the fault, and convergence is guaranteed. As a consequence, we may conclude that our findings hold for general fixed-point iteration schemes.

Explanation of the jumps. This behaviour can be explained by a smoothness argument. The Laplace operator (second directional derivatives) essentially describes the curvature, and even in early cycles, the error and thus the residual is already relatively smooth since the approximate solution is relatively smooth, cf. Figure 1. Our fault injection model (replacement of values) generally implies a weak singularity in the solution. This singularity translates to a steep change of curvature at the patch boundary, which explains the large jump, and also why the jump reaches the same plateau every time. The magnitude of the jump however corresponds to the mesh width h, the smaller h, the larger the jump. We can see in Figure 3, that the residual reaches a global maximum and a global minimum within one element layer at the patch boundary, corresponding to the two changes in curvature. Note that when injecting zeros, the residual is small inside the faulty patch, at least for small patch sizes and the smooth test problem. This however does not affect the jumps on the patch boundary, which dominate the failure. For other differential operators, e.g., the gradient in convection-diffusion problems, we expect similar behaviour.

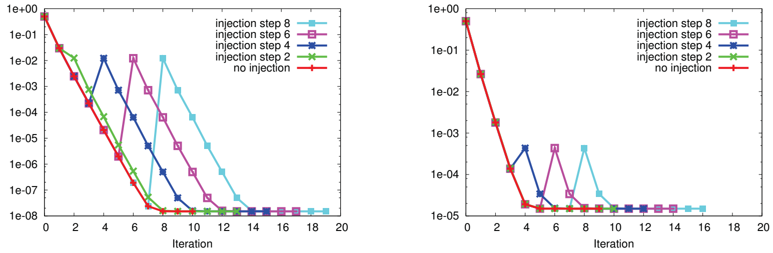

Figure 4: Multigrid convergence in the L^2 (y-axis) norm for the smooth (left) and oscillatory (right) test problem, see Figure 2 for the analogous curves in the residual norm.

Analysis in the error norm. Our analysis so far has focused on residual-based convergence control. To gain further insight, Figure 4 depicts the same experiment, but this time in the L^2 norm, see Section 3.1. We observe a similar jump in the error as in the residual norm, which here is directly caused by the weak singularity at the patch boundary. However, this jump and thus the fault injection only delay convergence if the fault dominates the L^2 error. This is particularly clear for the oscillatory problem: The solvers converge identically to the fault-free case as long as the fault is injected early enough, i.e., as long as the discretisation error does not dominate the overall error. For instance, an injection after the second cycle has no negative impact on convergence, and the jumps in the error are less pronounced for later injections. The mesh resolution we use translates to different 'plateaus' at which the discretisation error can no longer be reduced, depending on the smoothness of the problem. For the smooth test problem, the discretisation error is approximately 10^{-8}, and for the oscillatory problem, it is three orders of magnitude larger. As the order of the additional error caused by the fault is roughly the same in both test cases, the jumps in the error are (almost) equally pronounced as in the residual norm for the smooth case, and less dominant for the latter case.

In our implementation, the solver iterates until a residual-based convergence criterion is met, while the error is not reduced further once it has reached the level of the discretisation error. Our results indicate that ultimately, convergence control based on error estimators rather than the residual [39] may result in overall less work, as faults in later cycles would simply not happen with such a convergence criterion, cf. the magenta and cyan plots (injection steps 6 and 8) in Figure 4 on the right.

3.4. Recurrent faults

To conclude this first set of experiments, and to motivate our hierarchical asynchronous local failure local recovery approach, we consider the case of recurrent fault injections, exemplarily for the smooth test problem. We randomly pick small patches after every third cycle, and set the fine-grid solution values in these patches to zero. A single patch again corresponds to 0.02 % of the entire solution. Figure 5 depicts the resulting convergence of the multigrid solver. As expected from our previous analysis, the solver fails to converge while the fault injector is active, in both the residual and the error norm: Convergence is not fast enough to compensate for the large jumps caused by each fault. Thus, we may conclude that multigrid is only self-stabilising for single and sufficiently infrequent faults, but not for recurrent ones.

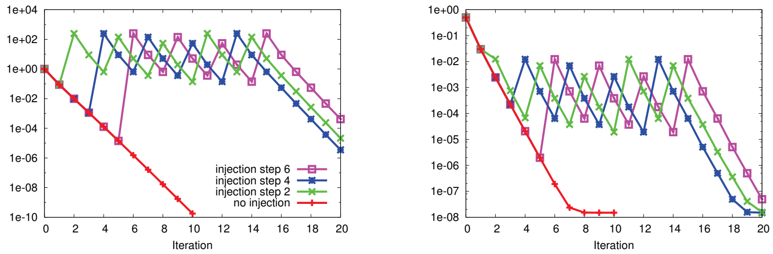

Figure 5: Convergence of the smooth test problem for injected faults at different locations after every third multigrid cycle, in the scaled residual (left) and L^2 (right) norm.

4. Fault-tolerant multigrid with hierarchical, asynchronous checkpointing

4.1. Motivation

Classical checkpoint-restart techniques are designed to be 'black-box', the user only specifies which state should be preserved. Checkpoints are taken periodically, and for consistency reasons, they are taken globally synchronously, e.g., after every other timestep in a transient simulation. In the traditional approach, the restart after a fault is also global and synchronous, because this is the most straightforward way to guarantee consistency. A lot of research has been devoted to improve classical checkpoint-restart, including in particular efforts towards scalable uncoordinated checkpointing protocols, see Section 2 for examples.

There is general consensus that global synchronous techniques are too slow and their data volume is too high, in particular to increase robustness and reliability of (linearised) subproblems which only constitute one, yet often the most time-consuming, component in real simulations. Our proposed approach, which we present in Section 4.2, addresses all three issues, i.e., data volume, synchronicity, and globality, albeit differently and in particular in a more problem-specific way than the general solutions mentioned above. See Section 1.3 for our argument why this can be beneficial.

4.2. Proposed approach

We describe our novel fault-tolerance approach for multigrid methods by starting from the 'classical' coordinated globally synchronous checkpoint-restart paradigm, because we believe that this makes our ideas easier to understand. In essence, we propose to checkpoint at more fine-granular intervals, which is enabled by substantially less overhead and load imbalance in the recovery case compared to existing solutions.

The main idea of our approach is to exploit the existing grid hierarchy and thus the data compression that is already built into the algorithm, albeit for other purposes. Even though we focus on geometric multigrid solvers in this paper, we believe that similar techniques can be devised for other algorithms that rely on some sort of hierarchical information representation, such as hierarchical basis techniques used in multiscale methods, or wavelet compression. Our approach follows the 'local failure local recovery' paradigm, cf. Teranishi and Heroux [15].

Checkpointing. As we (currently) assume that all data, i.e., discrete operators on all multigrid levels and the right hand side on the finest level, are not affected by faults, it suffices to periodically checkpoint the fine-grid solution only, after each fault-free iteration: The matrices and the right hand side are constant during a linear solve, and all remaining auxiliary arrays of the multigrid algorithm carry no useful information, because we consider the cycle as our level of granularity.

Since a fine-grid array may be large, we reduce the data volume by storing only a coarser representation. Similar to the 'full multigrid' algorithm (FMG, see [1, 2, 14]), we use a restriction operation to construct a suitable checkpoint. In this work, we employ unweighted injection as restriction, i.e., we keep all solution values in DOF locations that are common to a finer and a coarser grid, and discard all other values. The restricted array is smaller by a factor of four in 2D, and a factor of eight in 3D, per level of coarsening. Overall, we obtain a geometric reduction in data volume and thus substantial savings. The target level of the restriction process is a free optimisation parameter, which we investigate in Section 4.3.

Since the checkpoint is taken directly after convergence control, our approach does not require an additional global synchronisation point. Instead, we can copy the current fine-grid iterate, take the checkpoint asynchronously to the computation of the next multigrid cycle using purely local, independent computations, and also store it asynchronously. Ignoring communication, the cost of the checkpoint creation is smaller than the cost of the down-sweep of a V-cycle. This bound is very conservative, since we do not apply smoothing during the restriction phase of the checkpoint creation, and since the injection procedure corresponds to a simple copy. We conclude that in our scheme, the fault-free performance is barely reduced. We discuss the performance and overhead in more detail in Section 5.2.

Recovery. When data loss is detected, we take the restricted checkpoint, and prolongate it back to the fine grid. This is not done globally, but only for the portion of the checkpoint that corresponds to the faulty patch (the lost node), plus eventually one or two additional layers of neighbouring values depending on the size of the support of the prolongation operator. We may thus refer to our scheme as a local recovery. Again, any prolongation operator except injection may be used, and in this work, we employ the finite element interpolation we already use for the multigrid. In our tests, we found that in contrast to full multigrid, it is not necessary to use higher-order interpolation. The extra costs of the recovery are again substantially less than the up-sweep of a V-cycle. Similar to the FMG scheme, we may also apply some smoothing locally.

It is important to note that we perform the recovery instantaneously, and do not execute multigrid cycles between detection and correction. However, the checkpoint may be asynchronous, i.e., it does not necessarily have to be based on the very last fault-free cycle but may be older. We examine the influence of the degree of asynchronicity experimentally in Section 4.3.

While the backup solution is restored, the solver may make progress elsewhere. In practice, this typically means that all MPI processes that are no direct neighbours may continue, because data exchange in parallel multigrid methods is limited to immediate neighbours, see Section 1.1. This holds until the next global synchronisation point is reached, which in current implementations of parallel multigrid typically means convergence control, or the solution of the coarse-grid problem. To alleviate the potential load imbalance, we suggest to execute a cheaper, numerically weaker smoother during the next down-cycle on the patch after recovery (on all other patches, the original smoother is employed), so that all processes converge again no later than at the coarse-grid solver. In fact, temporarily reducing the amount of presmoothing only on the fine grid should be more than sufficient in practice.

4.3. Numerical evaluation for single faults

In this section, we analyse the convergence of the proposed solver. Unless otherwise noted, the fault injection and solver configuration are unchanged from the previous tests in Section 3.3. Building upon our findings in Section 3, we initially restrict ourselves to two representative scenarios, corresponding to a fault injection into an ‘early’ cycle, i.e., no convergence yet in neither the residual nor the error norm, and a ‘late’ cycle, i.e., convergence not in the residual but in the error norm (for the refinement level we consider). For the smooth problem, the ‘early’ scenario corresponds to a fault injection and the immediate recovery after the fourth cycle, and the ‘late’ scenario after the eighth cycle.

In these tests, the main free parameter is the compression factor of the checkpoint, measured in multigrid levels (backup depth) starting from the fine grid at depth zero. A backup level of 0 corresponds to no compression, and for each additional coarsening level, the compression rate increases by a factor of four, since we solve 2D problems. Unless otherwise noted, the checkpoint corresponds to the previous cycle, respectively. We again emphasise that in these experiments, we are only interested in the numerical behaviour of fault-tolerant multigrid algorithms.

Early scenario. Figure 6 depicts the results we obtained for the smooth test problem and the ‘early’ scenario, i.e., we inject and immediately recover from a fault when the solution is neither converged in the residual nor the error norm. In the residual norm, we again observe a jump despite the recovery, which is however substantially smaller than in the uncorrected case. For moderate compression rates (backup depths up to four levels, i.e., a factor of 256, the solver requires only one additional iteration, and even for strong compression, only two extra iterations are needed: The cyan (filled squares) plot corresponds in 2D to a compression factor of $4^6 = 4096$, i.e., the full, global checkpoint comprises only 289 out of 1 050 625 unknowns. Depending on the location of the faulty patch, only one unknown in the compressed backup represents the entire 13×13 patch. These savings are remarkable, both in terms of data volume and iterations. In the L^2 norm (Figure 6 on the right), convergence is identical to the error-free case.

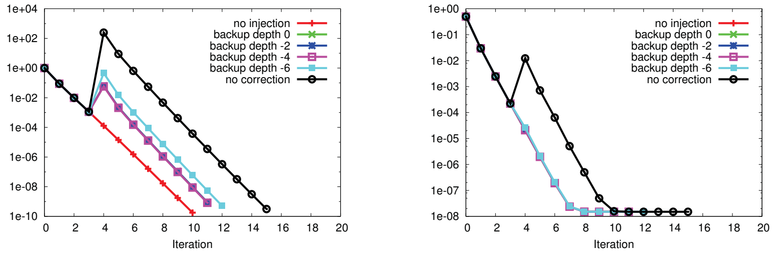

Figure 6: Convergence of the fault-tolerant multigrid solver in the scaled residual (left) and L^2 (right) norm, for the smooth test problem. After the fourth cycle, 169 degrees of freedom around the centre of the domain are eliminated and immediately recovered from a checkpoint of varying backup depth (compression level).

The behaviour is qualitatively and quantitatively the same for the oscillatory test problem, see Figure 7: As expected, strong compression has diminishing returns, because the relevant error components cannot be represented on too coarse grids.

Late scenario. Figure 8 shows the analogous situation for the ‘late’ scenario and the smooth test problem. For the same moderate compression factors, we

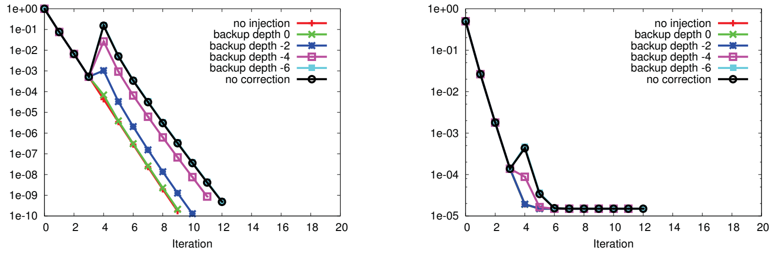

Figure 7: Convergence of the fault-tolerant multigrid solver in the scaled residual (left) and L^2 (right) norm, for the oscillatory test problem. After the fourth cycle, 169 degrees of freedom around the centre of the domain are eliminated and immediately recovered from a checkpoint of varying backup depth (compression level).

observe similar behaviour as in the previous scenario: In the residual norm, the amount of cycles until convergence mildly increases, and in the L^2 norm, convergence is identical to the fault-free case. If however the compression is too strong (more than four levels, i.e., more than a factor of 256 in 2D), the recovered values on the patch are increasingly less beneficial. In both norms, we again observe a jump. Its size increases with the compression level, resulting in a proportional increase in iterations until convergence. In the oscillatory case, the compression is less beneficial for the mesh resolution we prescribe, see Figure 9. However, the uncompressed previous iterate fully restores the behaviour of the fault-free case.

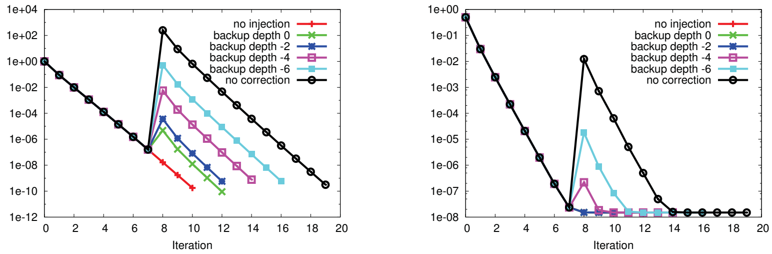

Figure 8: Convergence of the fault-tolerant multigrid solver in the scaled residual (left) and L^2 (right) norm, for the smooth test problem. After the eighth cycle, 169 degrees of freedom around the centre of the domain are eliminated and immediately recovered from a checkpoint of varying backup depth (compression level).

Detailed analysis. Due to space constraints, in the remainder of the paper we only examine the smooth test problem, because as we have seen, there are no important qualitative differences. Based on our findings in Section 3, these jumps are in fact the expected behaviour: The purely local restoration again results in a weak singularity along the boundary of the faulty patch, this time in the corrected iterate. Without compression, the recovery replaces the current (faulty) iterate with the one from the previous cycle in the faulty patch. This can be clearly seen in Figure 10 (left), where we observe that the recovery (for this test case) results in a small overshoot in parts of the patch: The exact solution value in the centre of the domain is minus one, and the prolongated previous iterate takes the value -1.04. With increasing compression, the severity

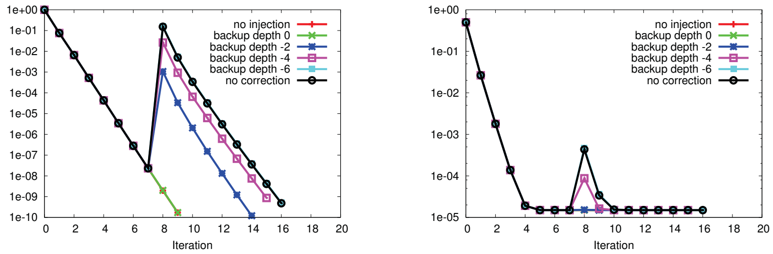

Figure 9: Convergence of the fault-tolerant multigrid solver in the scaled residual (left) and L^2 (right) norm, for the oscillatory test problem. After the eighth cycle, 169 degrees of freedom around the centre of the domain are eliminated and immediately recovered from a checkpoint of varying backup depth (compression level).

of this weak singularity gradually approaches the uncorrected case.

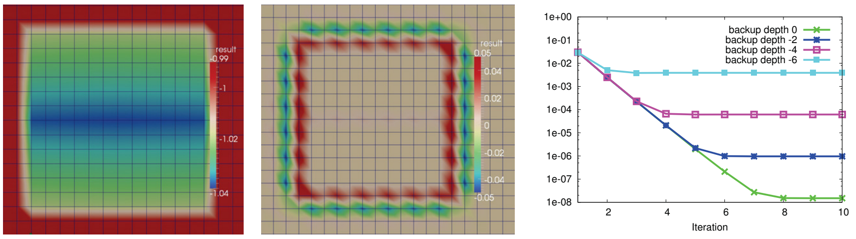

Figure 10: Impact of the recovery for the smooth test problem: Zoom on the iterate (left) and the residual (middle) after correcting the iterate without compression. One finite element cell corresponds to four cells in the illustration, i.e., each grid point corresponds to one degree of freedom. Right: L^2 norm of the compressed checkpoint for different backup levels at different iterations.

In the residual, this singularity again translates to a sharp change in curvature, which in turn manifests itself as a jump. See Figure 10 (middle) for an illustration. Importantly, for moderate backup levels up to four (corresponding to a compression by 256) for the smooth case, this jump is substantially smaller than in the uncorrected case, and overall behaves proportionally to the backup level.

In the error norm, the quality of the recovered solution can be quantified. To illustrate this, we compute the L^2 norm of the compressed checkpoint at different cycles and backup levels, see Figure 10 (right). Without compression, the error reduction is in line with the fault-free case, cf. the green plot in Figure 10 (right) with the red and green plots in Figures 6 and 8. The more compression (backup depth) we apply, the earlier the error saturates, both in terms of iterations and absolute amount. This is the expected behaviour, because with increasing iterations, more and more remaining error frequencies can only be captured on finer and finer grids. The progress that the solver has made so far is already optimal with respect to the mesh width, and a too coarse checkpoint resolution results in stagnation of the error reduction, because the achievable discretisation error dominates. Figuratively speaking, the correction with the recovered solution from the previous cycle is always the same from a certain iteration onwards.

A brief derivation based on the weak formulation (see AppendixA) yields the following general quantitative estimate for the case of fault injection into an already converged solution in the L^2 sense. If we denote by γ the fraction of unknowns that comprise the faulty patch, then

$$J_k \sim 8^k \sqrt{\gamma} \tag{2}$$

holds for the expected jump J_k in the error norm and biquadratic conforming elements, where k denotes the difference between the finest and the backup level. The square root stems from the use of the L^2 norm. In our case, $\gamma \approx 0.02\%$, and the estimate is well in line with the quantitative observations: For instance, in Figure 8 (right), and $k = 6$, the formula yields approximately $3.3 \cdot 10^3$, and the jump in the cyan plot ('backup depth -6') is approximately $1.2 \cdot 10^3$.

These observations suggest that the backup level should be chosen depending on the progress that the solver has achieved so far. We return to this question in Section 4.4.

Asynchronous checkpoints. One of the advantages of our approach is that the checkpoints can be taken asynchronously, or more precisely, that any available previous checkpoint can be used as a starting point to prolongate the correction term in the faulty patch. To evaluate the impact of older values numerically, we consider the smooth test problem and the same single fault injection as throughout this section, and exemplarily a backup depth of four, i.e., a compression factor of 256.

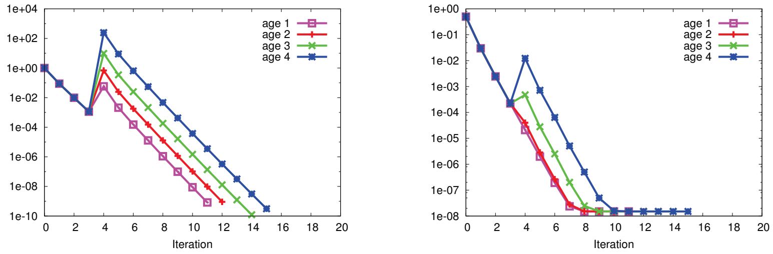

Figure 11: Impact of the age of the checkpoint (measured in cycles) for the smooth test problem and a backup four levels coarser. Fault injection after the fourth cycle, scaled residual (left) and L^2 (right) norm.

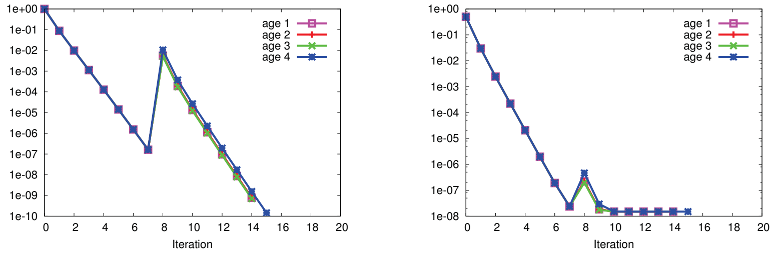

Figure 12: Impact of the age of the checkpoint (measured in cycles) for the smooth test problem and a backup four levels coarser. Fault injection after the eighth cycle, scaled residual (left) and L^2 (right) norm.

Figures 11 and 12 depict the convergence behaviour of the fault-tolerant multigrid solver for increasing age, measured in cycles, of the checkpoint, for the 'early' and 'late' scenarios. An age of one corresponds to the checkpointed iterate from the previous cycle, and thus to the situation in our previous tests. For an early fault injection, we observe a gradual increase in the number of iterations needed until convergence, caused again by an increasing jump in the residual. Since in practice we only expect one or two cycles delay, the asynchronicity is paid for with one or two additional iterations. In the L^2 norm, only large delays cause visible differences. In the late scenario, the delay has virtually no effect at all, in both norms. This is a direct consequence of the fact that at this compression level, the stored checkpoint is too coarse to capture the remaining error frequencies, cf. our detailed analysis above and Figure 10 (right).

Discussion. If we consider conventional convergence control based on residual norms only, our fault-tolerance mechanism for multigrid requires less iterations compared to the uncorrected case in all our experiments. This holds as long as the compression level is not chosen extremely. Only for very late faults, i.e., in the case that the solution is already converged in the L^2 sense, and for aggressive compression, the amount of iterations approaches but never exceeds the uncorrected case. This, along with the other advantages we discussed, is an important benefit of our proposed approach.

It is instructive to compare the resulting iteration numbers with the ones that a global checkpoint-restart mechanism would yield. If we detect the error and immediately globally restart the solver with the original initial guess, we would require at most twice the amount of iterations. Our solver always requires substantially fewer iterations if the compression level is not chosen too aggressively, and does not require any global restart.

If, on the other hand, we would always checkpoint the current iterate and restart from it, we would need exactly one additional iteration. This tradeoff is actually a major advantage of our proposed method: It allows to balance the cost of a global checkpoint-restart scheme (which is a special case of our method) with the additional iterations implied by our local recovery scheme. Given that our approach yields the same behaviour as the fault-free case if the compression depth is not chosen too ambitiously, there is ample head space to explore in architecture and problem-size specific tuning. We return to performance considerations in Section 5.2.

4.4. Recurrent faults

In Section 3.4 we have observed that the multigrid solver fails to converge in the presence of recurrent faults. To examine if our recovery mechanism alleviates this issue, we consider the case of fault injection after every third cycle, starting from the second cycle and continuing without faults at the 16th cycle. Figure 13 depicts the convergence plots in the residual and error norms, respectively.

We can see that our fault-tolerance scheme is not able to guarantee convergence in the residual norm. This is actually not surprising, given our previous analysis: After the error has been reduced to a certain order, the checkpoint does no longer contain any uneliminated error components (frequencies), and convergence stalls and jumps appear. In this sense, the plot corresponding to not applying any compression (green, filled squares) is actually misleading, if we would have injected faults more often, the solver would have failed to converge as well. With increasing compression level, the situation gets worse. It is instructive to compare this figure with the quality of the checkpoint in the

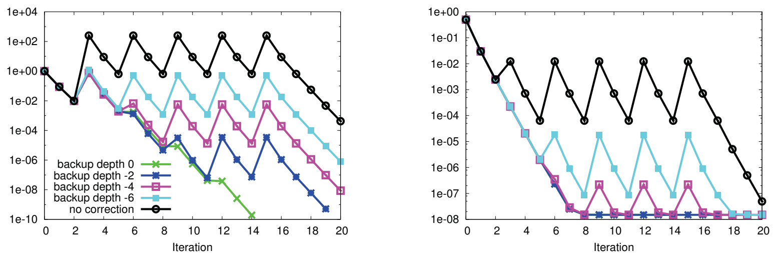

Figure 13: Impact of the recurrent fault injection and immediate recovery from varying compression levels, for the smooth test problem: scaled residual (left) and L^2 (right) norm

L^2 sense, shown in Figure 10 (right). In addition, the injection frequency is (deliberately) too high in this test case, so that the solver fails to compensate for the jumps in the fault-free cycles.

In the L^2 norm however, we observe convergence to the order of the discretisation error for moderate compression levels, i.e., up to two levels of compression. This again underlines that convergence control based on error estimators rather than the residual may be beneficial despite the additional numerical effort, see the end of Section 3.3.

Improving convergence. To achieve convergence even in the residual norm, we have to abandon the idea of recovery based on replacing lost patches with information from the previous iteration only. Instead, we apply a standard idea for elliptic boundary value problems: Instead of replacing lost values with previous ones, we communicate the fault-free layer of values from the immediate neighbourhood of the lost patch to the lost node, and use them as Dirichlet data for an auxiliary problem on the patch. If we then solve this small auxiliary problem, we recover the entire lost patch and thus guarantee convergence of the full problem. This approach is somewhat similar to Laplace corrections originally proposed in image processing [40], and has also been discussed for the GMRES method by Langou et al. [41].

From an implementation point of view, these local solves may be cumbersome. In our FEAST framework (see Section 1.1 and reference [6]), such local solves are actually an integral part of the patch concept and the solution strategy. Even though such local solves, in particular with a local multigrid method, have been integrated for numerical scalability reasons in case of strong anisotropies and locally varying coefficients, this infrastructure is now very beneficial, since we can employ it (almost) without additional effort.

Adaptive local solves. Even though this auxiliary problem is small compared to the global large-scale problem, solving it may be costly. Also, in our envisioned scenario of a dead node, the local problem is so large that sparse direct solvers are inefficient both in terms of runtime and additional memory, and we have to resort to an iterative solver. We discuss several strategies to reduce the impact of the local solve, which in turn reduce the implied load imbalance of the global solver. In our experiments, we found that simply applying the smoother for a small number of local iterations is not beneficial. Even though it helps to reduce the jumps in the case of early faults, we ultimately reach the case where smoothing alone, on the fine grid, does not reduce the error frequencies fast enough that remain in the iterate. In fact, in our experiments we obtain results qualitatively identical to Figure 13.

We thus have to solve this auxiliary local problem ‘exactly’. As a first adaptivity criterion, we note that it suffices to prescribe a (residual-based) stopping criterion that corresponds to the progress that the outer solver has reached so far, eventually equipped with a small safety margin. In other words, the later an error occurs, the more exact must the auxiliary problem be solved. This strategy is obvious.

Our second adaptive scheme makes use of the local compressed backup. Even though it is not useful on its own as a checkpoint, it is extremely useful as an initial guess for the local solve. Thus, all outlined benefits of our approach in terms of data volume, locality and asynchronicity remain valid. Figure 14 depicts on the left the convergence of our final fault-tolerant multigrid scheme, for varying compression levels of the initial guess for the local auxiliary solve, in the same case of recurrent faults examined above.

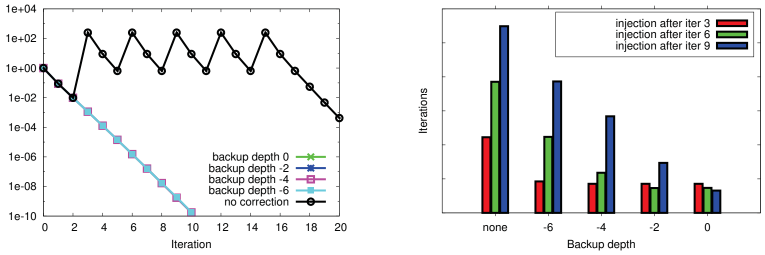

Figure 14: Convergence of the final fault-tolerant multigrid equipped with local solves using the (compressed) checkpoint as an initial guess, in case of frequent, recurrent faults. Left: convergence in the scaled residual norm. Right: Iterations of the local solver at different fault injection cycles, for varying backup depths (compression levels); ‘none’ denotes an initial guess of all zeros instead of the checkpoint. As the implementation is prototypical (we employ the smoother as a local solver), the y axis is unlabelled and we do not argue about the amount of savings in iterations. The relative differences however are independent of the type of the local solver.

Looking at the iterations of the local solve given in Figure 14 on the right, we first observe that, as expected, solving the local problem requires more and more iterations the later a fault occurs. Using the checkpoint from the previous cycle as an initial guess is always beneficial. The numbers also highlight that choosing the compression level adaptively in the course of the global solution is a good strategy. For instance, an aggressive compression yields the same iterations as no compression for early faults, etc. Ignoring small granularity effects, we can state that there is always an optimal compression level that yields iterations comparable to those of the uncompressed checkpoint. This relation is linear and can thus be easily implemented heuristically.

5. Practical aspects and performance modelling

5.1. General discussion

As outlined in Section 1.3, one of the most common scenarios of ‘data loss’ in multigrid methods is when an entire compute node dies. Our notion of a ‘node’ is deliberately vague, but without loss of generality we can restrict the discussion to the case that in some MPI communicator, at least one rank is unresponsive. We briefly explain how such a situation can be accounted for with recently proposed additions to the MPI standard.

In particular, so-called ‘revoke’ mechanisms for MPI communicators involving nonresponsive (presumably dead) ranks seem good candidates. Research in this direction is pursued for instance in the FT-MPI project [42]. Also, we anticipate this or similar techniques to be included[1] (in one way or another) in the upcoming MPI-4 standard, and thus to become widely available along with the arrival of actual exascale machines. Essentially, the FT-MPI proposal only specifies a general API providing mechanisms to gracefully survive hard and soft failures, centred around the ULFM (user level failure mitigation) paradigm, i.e., a set of MPI interface extensions to enable MPI programs to restore MPI communication capabilities disabled by failures. For instance, $O(\log p)$ algorithms are provided (where p is the number of processes) to determine which rank has failed. Using such functionality, the actual mitigation including the spawning of a new MPI process must be implemented by the application. In the case of our proposed resilient multigrid solver, this means that for instance, values from neighbouring nodes need to be communicated. But since the required information is essentially already available from the way parallel multigrid solvers are implemented, we anticipate that not much more than a few `allreduce`-type operations on the neighbourhood communicator need to be added.

5.2. Performance and overhead estimation

To model the performance and benefits of our approach, we assume a macro mesh (see Section 1.1) where each patch is assigned to one processor. Each patch is uniformly refined L times into an $N \times N$ mesh. The numbering starts with zero for the coarse mesh, and level L for the fine grid. We model a ScaRC multigrid solver, with local multigrid per node and communication only via patch boundaries (ghost layer of one element).

In our model, we are primarily interested in the time budget that our approach yields, and in the time compared to classical synchronous checkpointing. Hence, we ignore the case where the solution of an auxiliary problem is necessary. We use the following quantities:

t_{op}	-	time for one floating point operation
α	-	latency of communication between nodes
BW	-	bandwidth between nodes
n_{smooth}	-	sum of pre- and postsmoothing steps
p	-	order of the FEM discretisation, i.e., Q_p
δ	-	additional iterations due to the fault
depth	-	depth of the backup solution

We restrict our model to the 2D case for simplicity. For a finite element ansatz with Q_p elements, pN is an approximation of the number of degrees of freedom per spatial dimension: $Q_1 : N$, $Q_2 : 2N-1, \dots$). Furthermore, we assume double precision floating point arithmetics.

To estimate the computational cost, we need to know the number of nonzero entries of each matrix row. For conforming finite elements of order p, the support of each ansatz function comprises on average $(p+2)^2$ functions. For matching

[1] http://www.open-mpi.org/papers/sc-2014

prolongations and restrictions, the size of the support is bounded by the number of degrees of freedom per element, i.e., we obtain a support of at most $(p+1)^2$ functions. This simplification ignores prolongations of locations on edges, but since all inner points of an element require the full support, our approximation improves with increasing order.

The total runtime of our approach, including regular backups and one repair step, can then be expressed as

$$T_{\text{total}} = (n_{\text{iter}} + \delta)\left(T_{\text{smooth}} + T_{\text{cycle}} + T_{\text{backup}} + \left(\alpha + 8pN\frac{8\ \texttt{byte}}{\text{BW}}\right)\right) + T_{\text{repair}},$$

where we ignore the (tiny) time required by the coarse grid solves. The latency-bandwidth summand models time for all communications to patch neighbours. The time for smoothing,

$$T_{\text{smooth}} = \sum_{i=1}^{L}\left(\frac{pN}{2^{L-i}}\right)^2 \left(n_{\text{smooth}} \cdot (2(p+2)^2 + 2) + 2(p+2)^2 + 1\right) t_{\text{op}}$$

comprises the operations for pre- and postsmoothing, and the time needed to compute the residual prior to the restriction step. Here, we assume a damped Jacobi smoother, i.e., per smoothing step on average $2(p+2)^2$ operations are performed per degree of freedom for the matrix-vector multiplication and an additional two operations for the actual damped scaling by the matrix diagonal.

The prolongation takes

$$\sum_{i=1}^{L}\left(\frac{pN}{2^{L-i}}\right)^2 (p+1)^2\ t_{\text{op}}$$

time, and thus, the entire cycle without smoothing and coarse grid solution takes, assuming the transposed prolongation as restriction:

$$T_{\text{cycle}} = 2\sum_{i=1}^{L}\left(\frac{pN}{2^{L-i}}\right)^2 (p+1)^2\ t_{\text{op}}.$$

The construction of the backup is trivial, because we use the injection operator, and owing to the two-level numbering, only its leading entries have to be read. Thus, the amount of data needed for the backup is reduced exponentially in the backup depth, and we obtain:

$$T_{\text{backup}} = \alpha + \left(\frac{pN}{2^{\text{depth}}}\right)^2 \frac{8\ \texttt{byte}}{\text{BW}}.$$

The repair step contains the prolongation of the stored backup back to the fine grid. We employ the prolongation operator already used in the multigrid cycles:

$$T_{\text{repair}} = \sum_{i=L-\text{depth}+1}^{L}\left(\frac{pN}{2^{L-i}}\right)^2 (p+1)^2\ t_{\text{op}} + \alpha + \left(\frac{pN}{2^{\text{depth}}}\right)^2 \frac{8\ \texttt{byte}}{\text{BW}}$$

In contrast, classical checkpoint-restart techniques do not compress, yielding:

$$T_{\text{backup,CPR}} = T_{\text{repair,CPR}} = \alpha + (pN)^2 \frac{8\ \texttt{byte}}{\text{BW}}$$

Having all these estimates in place, we can now compute the time difference of the two methods, assuming that no additional iterations take place ($\delta = 0$). Obviously, the difference lies mainly in the costs for backup and repair. Note that positive values of the difference imply that our method is beneficial:

$$\begin{aligned}
& n_{\text{iter}} \left((pN)^2 \frac{8\ \texttt{byte}}{\text{BW}} - \left(\frac{pN}{2^{\text{depth}}} \right)^2 \frac{8\ \texttt{byte}}{\text{BW}} \right) && \text{(backup)} \\
& + (pN)^2 \frac{8\ \texttt{byte}}{\text{BW}} - \sum_{i=L-\text{depth}+1}^{L} \left(\frac{pN}{2^{L-i}} \right)^2 (p+1)^2\ t_{\text{op}} - \left(\frac{pN}{2^{\text{depth}}} \right)^2 \frac{8\ \texttt{byte}}{\text{BW}} \\
& && \text{(repair)} \\
= & \frac{n_{iter}+1}{4^{\text{depth}}} \left(4^{\text{depth}} - 1 \right) (pN)^2 \frac{8\ \texttt{byte}}{\text{BW}} - \frac{4(p+1)^2}{3} (1 - \frac{1}{4^{\text{depth}}}) t_{\text{op}}
\end{aligned}$$

For $t_{\text{op}} \ll \frac{1}{\text{BW}}$, we obtain the following upper bound for the savings in time, on top of the exponential savings in memory:

$$(n_{\text{iter}} + 1)(pN)^2 \frac{8\ \texttt{byte}}{\text{BW}} \tag{3}$$

Every additional iteration of the scheme, that is caused by the lossy compression, requires:

$$\begin{aligned}
& T_{\text{smooth}} + T_{\text{cycle}} + T_{\text{backup}} + \alpha + 8\ \frac{pN \cdot 8\ \texttt{byte}}{\text{BW}} \\
= & \sum_{i=1}^{L} \left(\frac{pN}{2^{L-i}} \right)^2 \Big(n_{\text{smooth}} \cdot (2(p+2)^2 + 2) + 2(p+2)^2 + 1 \Big) t_{\text{op}} && \text{(smoothing)} \\
& + 2 \sum_{i=1}^{L} \left(\frac{pN}{2^{L-i}} \right)^2 (p+1)^2 t_{\text{op}} + \alpha + 8pN \frac{8\ \texttt{byte}}{\text{BW}} && \text{(cycle+comm.)} \\
& + \alpha + \left(\frac{pN}{2^{\text{depth}}} \right)^2 \frac{8\ \texttt{byte}}{\text{BW}} && \text{(backup)} \\
= & \frac{4}{3} (1 - \frac{1}{4^L}) (pN)^2 \Big(n_{\text{smooth}} \cdot (2(p+2)^2 + 2) + 4(p+2)^2 + 1 \Big) t_{\text{op}} \\
& + pN (\frac{pN}{4^{\text{depth}}} + 8) \frac{8\ \texttt{byte}}{\text{BW}} + 2\alpha.
\end{aligned}$$

Application of the performance model. We exemplarily evaluate the model with a Q_2-FEM discretisation, i.e., $p = 2$ on a mesh hierarchy of eight levels ($L = 7$) and the four Jacobi smoothing steps ($n_{\text{smooth}} = 4$). As the target machine, we consider Hornet, as per the November 2014 TOP500 list Germany's third-fastest machine installed at the HLRS in Stuttgart. Hornet comprises 94 656 cores in 3 944 nodes.[2]. We scale the performance for typical sparse linear algebra computations of 39 TFLOPS[3] down to the node level, resulting in $t_{\text{op}} =$

[2] `http://www.hlrs.de/systems/platforms/cray-xc40-hornet`

[3] `http://www.hpcg-benchmark.org/custom/index.html?lid=155&slid=275`

0.101 ns. The available bandwidth between nodes is approximately 14 GBit/s (BW=1.75 GB/s), with a latency of $\alpha = 1\,500$ ns.[4]. We scale up the problem size to $(2 \cdot 2^{13})^2$ degrees of freedom, i.e., $N = 2^{13}$, matching the installed 128 GB of memory per node. Finally, we assume that in the fault-free case, the solver converges in $n_{\text{iter}} = 20$ iterations.

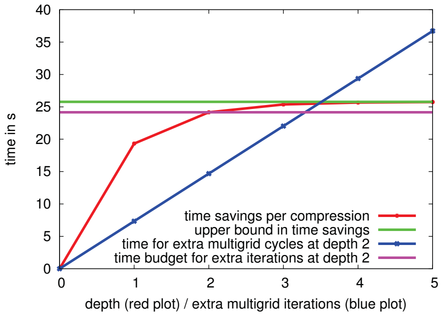

Figure 15: Exemplary model predictions of time savings and asymptotics

In Figure 15, we see exemplary results and predictions of our model. The green plot is simply the upper bound (3) for $t_{\text{op}} \ll \frac{1}{\text{BW}}$, i.e., the asymptotically maximum time available for additional iterations δ in our approach. Since we assume 20 iterations and the upper bound is predicted at 25 s, this means that our approach saves 25/20=1.25 s per iteration.

The red plot shows the time savings of our approach as a function of the compression depth. We observe that asymptotically, stronger compression becomes less effective, and quickly approaches the asymptotic limit.

We now take exemplarily the data point for depth = 2 from the red plot, this yields the magenta plot, showing the available time budget for this backup depth, again for 20 multigrid iterations. Note that this backup depth additionally corresponds to savings of a factor of eight in memory consumption. The blue plot shows the time for multigrid iterations (x-axis). Looking at the intersection of the two curves, we can see that within our model parameters, our model predicts a budget of $\delta \leq 3$ additional iterations to compensate for the lossy compression while still being beneficial compared to full checkpoints of the fine-grid iterate.

6. Outlook and future work

Our experimental analysis of the convergence behaviour and our performance model have shown noteworthy efficiency improvements of our approach to make multigrid solvers more resilient. In addition, the effort required by our family of techniques can be tailored to the expected mean time between failure.

There are several main avenues for future work. First, our experimental analysis needs to be backed up by theoretical considerations, as sketched in Section 3.3. Second, we need to consider other PDEs, for instance transport-dominated convection-diffusion problems, although we expect similar behaviour

[4] `https://www.nersc.gov/assets/pubs_presos/NUG2014Aries.pdf`

in the residual norm for gradients instead of the Laplace operator (curvature). Finally, we plan to experiment with other techniques, like improved interpolation, to dampen the impact of the weak singularity in and around the lost patch, that is the root cause of convergence degradation. Also, our prototypical implementation needs to be transferred to our FEAST code base in order to address scalability and load balancing issues associated with our approach and to assess how well our performance model works on real machines.

Acknowledgements

This work has been supported by the Mercator Research Center Ruhr (grant An-2013-0019), the German Research Foundation (Priority Programme 1648 'Software for Exascale Computing', grant Go 1758/2-1) and the 'Freundesverein der Fakultät für Mathematik, TU Dortmund'. All computations have been carried out on the LiDO$^{\text{ng}}$ supercomputer at TU Dortmund. We would like to thank Sven Buijssen, Stefan Turek and Peter Zajac for helpful discussions.

AppendixA. Derivation of the quantitative impact of fault injection on the error norm

In this appendix, we assume a finite element discretisation with biquadratic conforming elements, i.e., polynomial order $p = 2$. A simple linearity argument gives that

$$\left(\int_\Omega (u-u_h)^2 \mathrm{d}x\right)^{\frac{1}{2}} = \Big(\underbrace{\int_{\Omega\setminus\Omega_e} (u-u_h)^2 \mathrm{d}x}_{=O(h^{2(p+1)})} + \int_{\Omega_e} (u-u_h)^2 \mathrm{d}x\Big)^{\frac{1}{2}},$$

holds for the L^2 error, if Ω_e denotes the patch of lost data. The order-six term $h^{2(p+1)} = h^6$ is a direct property of the finite element space we use, see for instance [43]. We now replace u_h with a backup solution $u_{h,k}$, which is prolongated from a k-times coarser checkpoint of the previous fine-grid iterate. Here, analogously to Section 4.2, k denotes the level difference rather than an actual compression factor. Since we analyse the uniformly refined setting only, $k = 0$ corresponds to a checkpoint on the mesh h, $k = 1$ to a checkpoint for the mesh $\frac{h}{2}$ etc.

Substitution of the fully repaired solution $\tilde{u}_h$ yields:

$$\left(\int_\Omega (u-\tilde{u}_h)^2 \mathrm{d}x\right)^{\frac{1}{2}} = \Big(\underbrace{\int_{\Omega\setminus\Omega_e} (u-u_h)^2 \mathrm{d}x}_{=O(h^{2(p+1)})} + \underbrace{\int_{\Omega_e} (u-u_{h,k})^2 \mathrm{d}x}_{=2^{2k(p+1)}\ O(h^{2(p+1)})}\Big)^{\frac{1}{2}}$$

Assuming that the fraction $\frac{|\Omega_e|}{|\Omega|} =: \gamma$ is small, we obtain for biquadratic elements:

$$\begin{aligned}\left(\int_\Omega (u-\tilde{u}_h)^2 \mathrm{d}x\right)^{\frac{1}{2}} &\approx \left((1-\gamma)O(h^6)+\gamma 64^k\ O(h^6)\right)^{\frac{1}{2}} \\ &= (1-\gamma+64^k\gamma)^{\frac{1}{2}}O(h^3) \\ &\leq (1+64^k\gamma)^{\frac{1}{2}}O(h^3) \\ &\leq (1+8^k\sqrt{\gamma})O(h^3).\end{aligned}$$

This shows that the order of the jump, J_k, in the limit of a converged iterate, is proportional to

$$J_k \sim 8^k\sqrt{\gamma}$$

in the L^2 (error) norm.

References

[1] W. Hackbusch, Multi-grid methods and applications, 2nd Edition, Springer, 1985,2003.

[2] U. Trottenberg, C. W. Oosterlee, A. Schuller, Multigrid, Academic Press, 2001.

[3] U. Meier Yang, Parallel algebraic multigrid methods — high performance preconditioners, in: A. M. Bruaset, A. Tveito (Eds.), Numerical Solution of Partial Differential Equations on Parallel Computers, Vol. 51 of Lecture Notes in Computational Science and Engineering, Springer, 2006, pp. 209–236. doi:10.1007/3-540-31619-1_6.

[4] B. F. Smith, P. E. Bjørstad, W. D. Gropp, Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge University Press, 2004.

[5] S. Turek, D. Göddeke, C. Becker, S. H. Buijssen, H. Wobker, FEAST – Realisation of hardware-oriented numerics for HPC simulations with finite elements, Concurrency and Computation: Practice and Experience 22 (6) (2010) 2247–2265. doi:10.1002/cpe.1584.

[6] S. Turek, D. Göddeke, S. H. Buijssen, H. Wobker, Hardware-oriented multigrid finite element solvers on GPU-accelerated clusters, in: J. Kurzak, D. A. Bader, J. J. Dongarra (Eds.), Scientific Computing with Multicore and Accelerators, CRC Press, 2010, Ch. 6, pp. 113–130. doi:10.1201/b10376-10.

[7] J. Dongarra, et al., The international exascale software project roadmap, International Journal of High Performance Computing Applications 25 (1) (2011) 3–60. doi:10.1177/1094342010391989.

[8] D. E. Keyes, Exaflop/s: The why and the how, Comptes Rendus Mécanique 339 (2–3) (2011) 70–77. doi:10.1016/j.crme.2010.11.002.

[9] J. Dongarra, J. Hittinger, J. Bell, L. Chacón, R. Falgout, M. Heroux, P. Howland, E. Ng, C. Webster, S. Wild, K. Pau, Applied mathematics research for exascale computing, Tech. rep., U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research Program, http://science.energy.gov/~/media/ascr/pdf/research/am/docs/EMWGreport.pdf (Mar. 2014).

[10] D. Lammers, The era of error-tolerant computing, IEEE Spectrum 47 (11) (2010) 15–15. doi:10.1109/MSPEC.2010.5605876.

[11] J. Elliott, M. Hoemmen, F. Mueller, Evaluating the impact of SDC on the GMRES iterative solver, in: Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS'14), 2014, pp. 1193–1202. doi:10.1109/IPDPS.2014.123.

[12] M. Snir, et al., Addressing failures in exascale computing, International Journal of High Performance Computing Applications 28 (2) (2014) 129–173. doi:10.1177/1094342014522573.

[13] P. Sao, R. Vuduc, Self-stabilizing iterative solvers, in: Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA '13), 2013, pp. 4:1–4:8. doi:10.1145/2530268.2530272.

[14] L. Kronsjö, G. Dahlquist, On the design of nested iterations for elliptic difference equations, BIT Numerical Mathematics 12 (1) (1972) 63–71. doi:10.1007/BF01932674.

[15] K. Teranishi, M. A. Heroux, Toward local failure local recovery resilience model using MPI-ULFM, in: EuroMPI/ASIA '14, 2014, pp. 51:51–51:56. doi:10.1145/2642769.2642774.

[16] K.-H. Huang, J. A. Abraham, Algorithm-based fault tolerance for matrix operations, IEEE Transactions on Computers C-33 (6) (1984) 518–528. doi:10.1109/TC.1984.1676475.

[17] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, C. Engelmann, Combining partial redundancy and checkpointing for HPC, in: Proceedings of the 2012 IEEE International Conference on Distributed Computing Systems (ICDCS'12), 2012, pp. 615–626. doi:10.1109/ICDCS.2012.56.

[18] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, R. Brightwell, Detection and correction of silent data corruption for large-scale high-performance computing, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12), 2012, pp. 78:1–78:12.

[19] X. Ni, E. Meneses, N. Jain, L. V. Kalé, ACR: Automatic checkpoint/restart for soft and hard error protection, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13), 2013, pp. 7:1–7:12. doi:10.1145/2503210.2503266.

[20] T. Ropars, T. V. Martsinkevich, A. Guermouche, A. Schiper, F. Cappello, SPBC: Leveraging the characteristics of MPI HPC applications for scalable checkpointing, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13), 2013, pp. 8:1–8:12. doi:10.1145/2503210.2503271.

[21] R. Riesen, K. Ferreira, D. Da Silva, P. Lemarinier, D. Arnold, P. G. Bridges, Alleviating scalability issues of checkpointing protocols, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12), 2012, pp. 18:1–18:11.

[22] K. Sato, N. Maruyama, K. Mohror, A. Moody, T. Gamblin, B. R. de Supinski, S. Matsuoka, Design and modeling of a non-blocking checkpointing system, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12), 2012, pp. 19:1–19:10.

[23] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, S. Matsuoka, FTI: high performance fault tolerance interface for hybrid systems, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11), 2011, pp. 32:1–32:32. doi:10.1145/2063384.2063427.

[24] G. Bosilca, R. Delmas, J. Dongarra, J. Langou, Algorithm-based fault tolerance applied to high performance computing, Journal of Parallel and Distributed Computing 69 (4) (2009) 410–416. doi:10.1016/j.jpdc.2008.12.002.

[25] P. G. Bridges, K. B. Ferreira, M. A. Heroux, J. Elliott, Fault-tolerant linear solvers via selective reliability, Tech. Rep. 1206.1390, arXiv (Jun. 2012).

[26] P. Bridges, M. Hoemmen, K. Ferreira, M. Heroux, P. Soltero, R. Brightwell, Cooperative application/OS DRAM fault recovery, in: M. Alexander, et al. (Eds.), Euro-Par 2011: Parallel Processing Workshops, Vol. 7156 of Lecture Notes in Computer Science, Springer, 2012, pp. 241–250. doi:10.1007/978-3-642-29740-3_28.

[27] A. Bouras, V. Frayssé, Inexact matrix-vector products in Krylov methods for solving linear systems: A relaxation strategy, SIAM Journal Matrix Analysis and Applications 26 (3) (2005) 660–678. doi:10.1137/S0895479801384743.

[28] V. Simoncini, D. B. Szyld, Theory of inexact Krylov subspace methods and applications to scientific computing, SIAM Journal on Scientific Computing 25 (2) (2003) 454–477. doi:10.1137/S1064827502406415.

[29] T. P. Collignon, M. B. van Gijzen, Parallel scientific computing on loosely coupled networks of computers, in: B. Koren, K. Vuik (Eds.), Advanced Computational Methods in Science and Engineering, Vol. 71 of Lecture Notes in Computational Science and Engineering, Springer, 2010, pp. 79–106. doi:10.1007/978-3-642-03344-5_4.

[30] M. Hoemmen, M. A. Heroux, Fault-tolerant iterative methods via selective reliability, http://www.sandia.gov/~maherou/docs/FTGMRES.pdf (May 2011).

[31] A. Mishra, P. Banerjee, An algorithm-based error detection scheme for the multigrid method, IEEE Transactions on Computers 52 (9) (2003) 1089–1099. doi:10.1109/TC.2003.1228507.

[32] P. K. Jimack, M. A. Walkley, Asynchronous parallel solvers for linear systems arising in computational engineering, Computational Technology Reviews 3 (2011) 1–20. doi:10.4203/ctr.3.1.

[33] M. Huber, B. Gmeiner, U. Rüde, B. Wohlmuth, Resilience for exascale enabled multigrid methods, Tech. rep. (Jan. 2015).

[34] B. Schroeder, E. Pinheiro, W.-D. Weber, DRAM errors in the wild: A large-scale field study, in: SIGMETRICS/Performance '09: Proceedings of the 2009 Joint International Conference on Measurement & Modeling of Computer Systems, 2009, pp. 193–204. doi:10.1145/1555349.1555372.

[35] V. Sridharan, D. Liberty, A study of DRAM failures in the field, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12), 2012, pp. 76:1–76:11.

[36] J. Elliott, F. Mueller, M. Stoyanov, C. Webster, Quantifying the impact of single bit flips on floating point arithmetic, Tech. Rep. TR-2013-2, North Carolina State University (Mar. 2013).

[37] D. Li, Z. Chen, P. Wu, J. S. Vetter, Rethinking algorithm-based fault tolerance with a cooperative software-hardware approach, in: Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis, 2013, pp. 44:1–44:12. doi:10.1145/2503210.2503226.

[38] Y. Saad, Iterative Methods for Sparse Linear Systems, 3rd Edition, SIAM, 2003.

[39] R. Becker, C. Johnson, R. Rannacher, Adaptive error control for multigrid finite element methods, Computing 55 (4) (1995) 271–288. doi:10.1007/BF02238483.

[40] P. Pérez, M. Gangnet, A. Blake, Poisson image editing, ACM Transactions on Graphics 22 (3) (2003) 313–318. doi:10.1145/882262.882269.

[41] J. Langou, Z. Chen, G. Bosilca, J. Dongarra, Recovery patterns for iterative methods in a parallel unstable environment, SIAM Journal on Scientific Computing 30 (1) (2008) 102–116. doi:10.1137/040620394.

[42] W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bosilca, J. J. Dongarra, An evaluation of user-level failure mitigation support in MPI, Computing 95 (12) (2013) 1171–1184. doi:10.1007/s00607-013-0331-3.

[43] D. Braess, Finite Elements – Theory, Fast Solvers and Applications in Solid Mechanics, 2nd Edition, Cambridge University Press, 2001. doi:10.2277/0521011957.