NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2015; **00**:1–10
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nla

Resilience for Exascale Enabled Multigrid Methods

M. Huber[1*], B. Gmeiner[2], U. Rüde[2], B. Wohlmuth[1]

[1]*Insitute for Numerical Mathematics (M2), Technische Universität München,* [2] *Department of Computer Science 10, FAU Erlangen-Nürnberg*

SUMMARY

With the increasing number of components and further miniaturization the mean time between faults in supercomputers will decrease. System level fault tolerance techniques are expensive and cost energy, since they are often based on redundancy. Also classical check-point-restart techniques reach their limits when the time for storing the system state to backup memory becomes excessive. Therefore, algorithm-based fault tolerance mechanisms can become an attractive alternative. This article investigates the solution process for elliptic partial differential equations that are discretized by finite elements. Faults that occur in the parallel geometric multigrid solver are studied in various model scenarios. In a standard domain partitioning approach, the impact of a failure of a core or a node will affect one or several subdomains. Different strategies are developed to compensate the effect of such a failure algorithmically. The recovery is achieved by solving a local subproblem with Dirichlet boundary conditions using local multigrid cycling algorithms. Additionally, we propose a superman strategy where extra compute power is employed to minimize the time of the recovery process.

Received . . .

KEY WORDS: Fault Tolerance, Geometric Multigrid, Exascale, Massive Parallelism

1. INTRODUCTION

Future high performance systems will be characterized by millions of compute nodes that are executing up to a billion of parallel threads. This compute power will be expensive due to the associated investment and operational cost which will include energy consumption as a rapidly increasing factor. Hardware, software and algorithmic components of such a large scale computing are interdependent, and thus reliability of each single component is necessary. Since this increasing complexity results in a higher probability of any kind of failure in the HPC-system [1], strategies which circumvent and/or accomplish such a behavior are inevitable. A commonly used approach is copying the data to backup disks on the I/O-system and restart the run with the stored data in case of a failure. These checkpoint/restart strategies need to collect and transfer data from and to all processors and are, in general, too costly and not attractive.

Geometric multigrid methods can deliver an asymptotic optimal complexity and can be implemented with excellent efficiency on large scale parallel machines [2, 3, 4, 5]. Typical future runs involving multigrid computations will last from a few hours to weeks and use up to a billion threads. Therefore, an error-resilient methodology for any failure will be required and necessary to obtain fault-free and efficiently computed solutions.

The type of failures and their treatment can be categorized in hardware-based fault tolerance (HBFT) [6, 7, 8, 9], software-base fault tolerance (SBFT) [10, 11, 12, 13] and algorithm-based

*Correspondence to: huber@ma.tum.de

fault tolerance (ABFT) [14, 15, 16], for a general overview of fault tolerance techniques we refer to [1, 17, 18].

In this paper, we focus on the algorithmic approaches for handling failures. ABFT improves the reliability of the HPC-system in detecting the failure and correcting the results by implementing the resilience in the algoithm itself. Originally, ABFT was proposed by Huang and Abraham [14] for systolic arrays where due to checksums the persistency of the data involved in the algorithm is monitored and reconstructed. Later, it was extended to applications in linear algebra such as addition, matrix operations, scalar product, LU-decomposition, transposition and in fast Fourier transformation [19, 20, 21, 22]. Currently, the work by Davies and Chen [23] efficiently deals with fault detection and correction during the calculation for dense matrix operations. For iterative - linear and Krylov space - solvers for sparse matrix such as SOR, GMRES, CG-iterations the previous mentioned approaches are not suitable, since this can result in high overhead for sparse linear algebra [24] and were consequently adapted by [15, 25, 26, 27]. Cui et al. propose in [28] a technique to use the structure of a parallel subspace correction method such that the subspaces are redundant on different processors and the workload is efficiently balanced. Further, an algebraic multigrid solver was analyzed in [16] where the most vulnerable components are identified and resilience techniques are explored.

Here, we investigate in a fault tolerant parallel geometric multigrid method. Similar to [28] we pursue fault tolerant strategies which

- converge when a fault occurs assuming it is detectable,
- minimize the delay in the solution process,
- minimize computational and communication overhead.

In order to achieve these goals, we study the consequences of failures for a geometric multigrid algorithm from bottom up. By applying local correction methods, we recover a partial solution and use it such that the effect of the fault on the global solution procedure is minimized. The major difference to other approaches in ABFT [16, 28] is that we proceed without check-pointing data of the solution but rather recalculate the faulty part.

Our paper is organized as follows: In Sec. 2, we describe the model equation and briefly discuss the parallel hierarchical hybrid multigrid (HHG) framework that will serve as the basis of the study in this paper. Next, we introduce the failure scenario that is used to study the influence of a fault within a parallel geometric multigrid method and its effect on the convergence. In Sec. 3, we then develop local recovery strategies and demonstrate by numerical experiments how these improve the recovery behavior after the fault.

2. FAULTY SOLUTION PROCESS

2.1. *Model problem and geometric multigrid*

This paper considers, for simiplicity of notation, the Laplace equation with Dirichlet boundary conditions

$$-\Delta u = 0 \ \text{ in } \Omega, \qquad u = g \ \text{ on } \partial\Omega \tag{1}$$

as model problem for the design and analysis of a fault recovery algorithm. Here, $\Omega \subset \mathbb{R}^3$ is a bounded polyhedral domain. Ω is triangulated with an unstructured tetrahedral mesh that we denote by $\mathcal{T}_{-2}$. From this initial coarse mesh, a hierarchy of meshes $\mathcal{T} := \{\mathcal{T}_l,\ l = 0, \ldots, L\}$ is constructed by successive uniform refinement as illustrated in Fig. 1.

The discretization of (1) uses conforming linear finite elements (FE) on $\mathcal{T}_l$ that leads canonically to a nested sequence of finite element spaces $V_0 \subset V_1 \subset \ldots \subset V_L \subset H^1(\Omega)$ and a corresponding family of linear systems

$$A_l \underline{u}_l = \underline{f}_l;\ l = 0, \ldots, L. \tag{2}$$

The Dirichlet boundary conditions are included in the linear systems (2).

DOI: 10.1002/nla

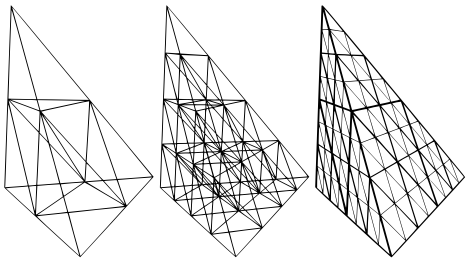

Fig. 1. Structured tetrahedral refinement.

This hierarchy will be used to set up an iterative multigrid solver and to define the error recovery strategy. Multigrid methods can achieve level-independent convergence rates with optimal complexity $\mathcal{O}(N)$, where N is the number of unknowns, cf. [29, 30]. We apply multigrid correction schemes in V-, W-, or F-cycles with standard components to (2). Explicitly, we use linear transfer operators and a hybrid variant of a Gauss-Seidel updating scheme as smoother.

2.2. *Hierarchical Hybrid Grids*

The Hierarchical Hybrid Grids (HHG) framework [2, 3, 4] is designed to combine the flexibility of unstructured FE meshes with the performance advantage of structured grids in a block-structured approach.

The implementation is based on domain partitioning that splits the mesh into primitives: vertices, edges, faces, and volumes. In the HHG data structure each primitive is then refined regularly resulting in a global block-structured grid. For our later error recovery strategies, the domain partitioning approach is crucial, but the block-structured mesh structure could be generalized to fully unstructured meshes. The multigrid operations such as smoothing, prolongation, restriction, and residual calculation, are exploited such that they typically operate on the primitive itself and its neighboring primitives via ghost layers. These operations are inherently local and suited for parallel computations on a distributed memory system using message passing with MPI. Here, the primitives are mapped to processors that execute the local operations. The data dependencies require a systematic exchange of the ghost layers. This functionality is provided in a transparent and highly optimized form in the HHG framework.

2.3. *Fault Model*

We assume that a failure in the solution process for (2) can occur at any time. For our study, we concentrate on a specific fault model under assumptions similar to [28, 31]. We restrict the analysis, for simplicity, to the case that only one process crashes. All strategies can be extended easily to a defect of more processors, since they only rely on the locality of the fault.

Furthermore, we concentrate on the case of using V-cycles for the solution of (2). The input tetrahedral mesh $\mathcal{T}_{-2}$ defines the partitioning used for parallelization in HHG. Each tetrahedron in $\mathcal{T}_{-2}$ is mapped to a processor, including all the refined subdomain meshes contained in the coarsest level element. Consequently, the number of subdomains and the number of processes is equal to the number of tetrahedra in $\mathcal{T}_{-2}$.

If a process experiences a fault, the information of the subdomain is lost. In the context of this article, the *faulty* subdomain $\Omega_F \subset \Omega$ is just a single tetrahedron in $\mathcal{T}_{-2}$. The other tetrahedra constitute the *healthy* subdomain Ω_H, i.e. those tetrahedra that are not affected by the fault. Healthy and faulty regions are separated by an interface $\Gamma_I := \partial\Omega_H \cap \partial\Omega_F$. In the finite element mesh, as implemented in HHG, the interface region Γ_I contains the nodes living on faces, edges, and vertices of the input mesh. These data structures are responsible for handling communication in HHG and are thus stored redundantly in the form of ghost instances on several processors. Thus, even if one of the instances is lost due to the fault, a complete recovery is always possible for them, and thus we assume implicitly that the data associated with them are unaffected by the fault.

DOI: 10.1002/nla

In Fig. 2, the setup is illustrated for a computational domain with 16 million unknowns. The domain consists of 48 tetrahedral subdomains that are distributed to 48 processors. Then each subdomain includes 300 000 unknowns and, thus, the failure of a process causes the loss of information for 300 000 unknowns.

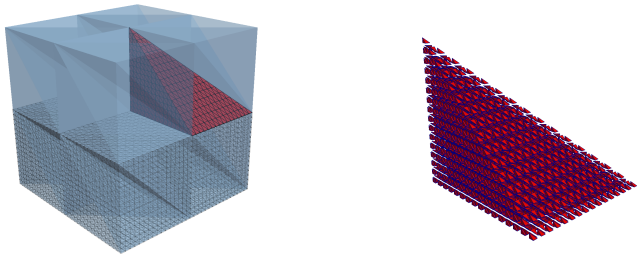

Fig. 2. Fault of one process. Left: Computational domain (here: 16 mil. unknowns) with a faulty (red) subdomain, one input grid tetrahedron (here: 300 000 unknowns). Right: Faulty subdomain.

For our strategy, it is necessary that we can detect erroneous processes quickly and then adapt the solution procedure dynamically. Unfortunately, the current supercomputer systems and the fault tolerant MPI-extensions such as Harness FT-MPI [†] or ULFM [‡] do not yet support this functionality as ideally needed. For the purposes of this study, we suppose a failure is reported as soon as it occurs during a multigrid cycle. When a process crashes, we assign a new - until then not used - substitute process to take over its job. This assumes that a large scale parallel computation is started with a certain number of substitute processors initially being idle – very much like substitute players in a team sport match. The solution values in the faulty subdomain are set to zero as initial guess. Other initial guesses could also be used, such as data from previous check-pointing or values obtained by interpolation, but this will not be considered here in detail. After the local re-initialization of the problem, we continue with multigrid cycles in the solution process.

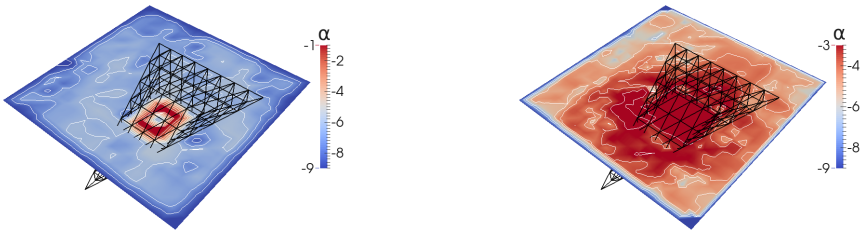

Fig. 3. Cross section through the domain Ω and the surface of the faulty tetrahedron. Left figure: residual error directly after the failure. Right figure: residual error after one additional global V-cycle. $\alpha := \log_{10}(|Residual|)$.

In a first experiment, we consider the performance of our multigrid iteration when it is continued after a fault has occurred. All numerical experiments are performed within the HHG framework introduced in Subsec. 2.2. We choose the computational domain $\Omega = (0,1)^3$, and $g = \sin(\pi(x + \sqrt{2}y))\sinh(\sqrt{3}\pi z)$ in (1) with the described setup in Fig. 2. In the solution process, we apply V-cycles with three pre- and post-smoothing steps of the Gauss-Seidel smoother of Sec. 2.1.

In Fig. 3, the residual is visualized on a cross section through the domain together with the surface of the tetrahedron where the fault had occurred after 5 global iterations. Right after the failure and after re-initialization, the largest residual error is clearly located in this tetrahedron. These large local error components are transported over the whole domain in the course of the

[†] http://icl.cs.utk.edu/ftmpi/
[‡] http://fault-tolerance.org/

DOI: 10.1002/nla

following multigrid iterations. Though each application of the smoother transports information only across a few neighboring mesh cells, multigrid employs coarser grids recursively. The smoothing on these grids leads to the global data exchange that is essential for the level-independent convergence of multigrid iterations. Therefore, though the residual is reduced efficiently (in the L_2-norm) by such iterations, we observe a pollution of the residual error across the whole domain. This can be seen in Fig. 3 at the right, where the overall residual error has been reduced after the additional V-cycle (note the scaling), but the error now pollutes the whole domain.

The numerical behavior is analysed quantitatively in Fig. 4. After the fault the residual norm jumps from $6.24 \cdot 10^{-7}$ up to $1.55 \cdot 10^{-1}$. If a *complete checkpointing-recovery* (CCR) of the lost values could be performed, it would fully restore the residual from before the fault. Note, that this recovery, as marked in the diagram with *no fault*, introduces no additional computational effort in comparison to the situation without failure, but writing and reading checkpoint data would be too expensive for large scale computations. However, the failure introduces error components that can be reduced efficiently by the multigrid method as can be seen in the residuals marked with *fault*. In the first cycles after the fault, we observe a pre-asymptotic convergence rate that is better than the asymptotic rate for roughly three cycles. This helps significantly to compensate for the fault. The roundoff error limit of approximately 10^{-15} is reached after a total of 20 V-cycles, as compared to 16 V-cycles that were necessary in the unperturbed computation. As expected these effects can be

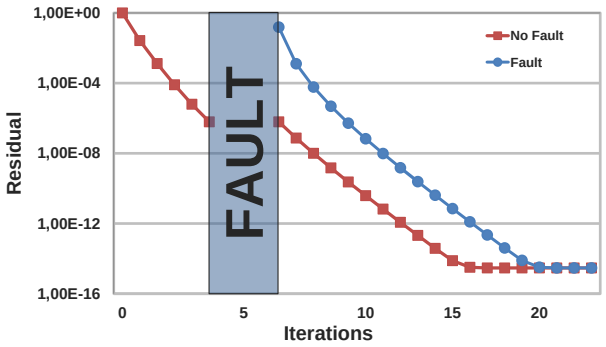

Fig. 4. Convergence of the residual error scaled by the initial residual with fault after 5 iterations.

seen more drastically, when the fault occurs at a later step during the iteration process. The situation of a fault after 7 iterations is displayed in Fig. 5 (left) and after 11 iterations in Fig. 5 (right). In those cases, the global residual is already quite small when the fault occurs, and we need 7 and 10 more iterations, respectively, to obtain the rounding error limit of 10^{-15}.

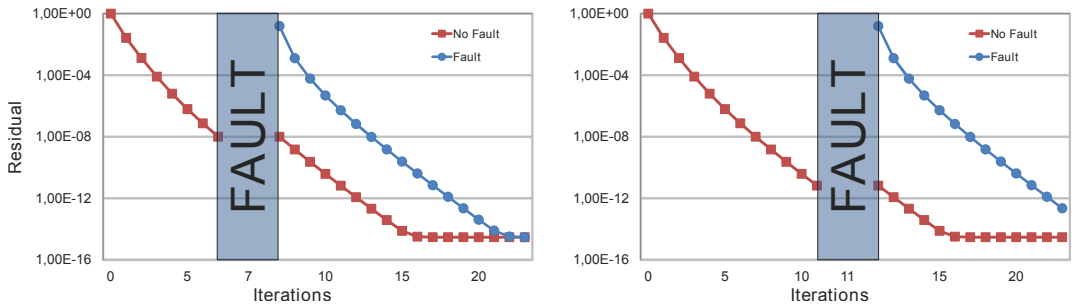

Fig. 5. Convergence of the residual error scaled by the initial residual with fault after 7 iterations (left) and 11 iterations (right).

DOI: 10.1002/nla

3. LOCAL RECOVERY STRATEGY

Avoiding the global pollution observed in Subsec. 2.3 motivates a recovery strategy. For the recovery step, we here require that it is local, i.e., can be performed without communication. For the overall efficiency of the recovery strategy, it is essential that it can be performed as quickly as possible. We therefore propose a *superman strategy*, which simply means that more resources are devoted to perform the recovery in an attempt to reduce the recovery time. This becomes especially attractive, since the recovery procedure is local, i.e. the *superman substitute processor* can focus its attention on a single subdomain. Note, that during the local recovery, the global solution process cannot continue unaffected, since this would require to use values from the faulty subdomain. This access to the faulty subdomain must be prevented, since otherwise a global error pollution would occur. The values in the faulty subdomain will only become available again, when the local recovery has been completed.

Technically the speedup of the superman strategy can be accomplished by additional parallelization. We propose here, that e.g. a full (shared memory) compute node is assigned to perform the local recovery for a domain that was previously handled by a single core. This can be accomplished by a dedicated OpenMP parallelization for the recovery process. Otherwise, of course, a further domain partitioning of the faulty region might be used together with an additional local distributed memory parallelization.

We denote the speedup factor that is achieved with a superman strategy for the local recovery by $\eta_{\text{speedup}} \in [1, \infty)$, i.e., if $\eta_{\text{speedup}} = 1$, there is no speedup. For the case $\eta_{\text{speedup}} \to \infty$, the recovery would cost no time. For the moment, let us assume such a *perfect superman* and that the local recovery step is in this sense free. Let us define the following local subproblem

$$-\Delta u = 0 \qquad \text{in } \Omega_F, \tag{3}$$

with Dirichlet boundary conditions on Γ_I.

Under the assumptions of Subsec. 2.3, we set the lost values in the faulty region $\Omega_F \subset \Omega$ to zero and before continuing with global problem (1) we solve (3).

The subproblem (3) can in principle be solved by any method, e.g., the relaxation that is used as multigrid smoothing, a direct solver, Krylov space iterations, or multigrid cycles. We denote by k_F the number of local solver iterations. After solving the subproblem (3) with sufficient accuracy, the solution process for the global problem (1) can be resumed. The procedure is presented in Alg. 1.

Algorithm 1 Local recovery algorithm

1: Solve (1) by multigrid cycles.
2: **if** Fault has occurred **then**
3: **STOP** solving (1).
4: Recover boundary values Γ_I by neighboring subdomains.
5: Set values of the defect subdomain Ω_F to zero.
6: Solve (3) in Ω_F subject to Dirichlet boundary conditions on Γ_I using
7: k_F local solver iterations with speedup factor η_{speedup}.
8: **RETURN** to step 1 with new computed values in Ω_F.
9: **end if**

To quantify the number of multigrid cycles that are saved by using the local recovery strategy, we introduce a reference parameter *Cycle Advantage* which we denote by κ. We assume a process experiences a fault after k cycles. Then, we evaluate the residual after K cycles with $k \leq K$. We choose K such that the stopping criteria is fulfilled in the no-fault case. Here, $r_K^{(k_F)}$ and $r_K^{(0)}$ denote the residual after a local recovery with k_F iterations and when no-local-recovery ($k_F = 0$) has been performed, respectively.

Then, it holds

$$\|r_K^{(k_F)}\| \leq \|r_K^{(0)}\|, \tag{4}$$

DOI: 10.1002/nla

since the local strategy improves the residual error. In the case of no-local-recovery, we need to apply additionally κ cycles such that

$$\|r_K^{(k_F)}\| = \|r_{K+\kappa}^{(0)}\| = \mu^\kappa \cdot \|r_K^{(0)}\|, \tag{5}$$

where μ is the convergence rate of one multigrid cycle. Thus, we solve for κ in (5)

$$\kappa = \log(\|r_K^{(k_F)}\|/\|r_K^{(0)}\|)/\log(\mu). \tag{6}$$

Note, that for the calculation of the reference parameter κ we need two versions of computation runs, one with a fault and a local recovery strategy and one with a fault and without a local recovery strategy. A local recovery strategy with $\kappa = 0$ does not improve the residual error in comparison to no recovery strategy, whereas a higher κ implies an improvement of the residual error of magnitude κ in multigrid cycles. For example, if $\kappa = 5$, then, the residual error without a local recovery needs 5 additional cycles to achieve the same residual error as the local recovery strategy.

Let us consider again the example of Subsec. 2.3. We study five different local recovery strategies: Gauss-Seidel smoothing (Smth), Jacobi preconditioned CG (PCG) iterations, local V-cycles, local W-cycles and local F-cycles.

In Fig. 6, we present three cross sections through the computational domain together with the surface of the faulty tetrahedron. In the left plot, the residual error directly after the failure is shown, in the middle plot the residual error after the local recovery with one F-cycle and in the right plot after an additional global V-cycle after the local recovery. We observe two major advantages of local recovery strategies compared to no-local-recovery: the local recovery reduces the residual error in the defective tetrahedron (middle plot of Fig. 6) and the error pollution is much smaller over the computational domain (right plot of Fig. 6).

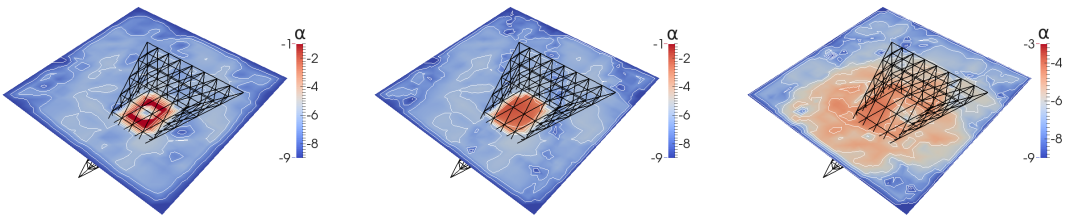

Fig. 6. Cross section through the domain Ω and the surface of the erroneous tetrahedron. Left plot: residual error directly after the failure. Middle plot: residual error after applying a local recovery strategy (one local F-cycle). Right plot: residual error after an additional V-cycle. $\alpha := \log_{10}(|Residual|)$.

In Fig. 7, we consider different local recovery strategies and their impact on the solution process. We observe that all local strategies improve the residual error directly after the fault in comparison to using no-local-recovery. The different cycles, local V- (green), local W- (purple), and local F-cycle (orange) strategies, produce almost the same effect on the residual errors. However, using 10 PCG-iterations (pink) or 10 Gauss-Seidel smoothing steps (light blue) result only in an almost negligible improvement compared to using no recovery at all. After the fault and recovery, all recovery strategies using a local multigrid solver exhibit a favorable pre-asymptotic convergence in the first three global iterations after recovery and, then, align with the convergence of CCR. The residual error of the smoothing or PCG strategy after three further iterations is similar to the approach without a local strategy and, therefore, needs as many iterations as the no-local-recovery strategy to reach the prescribed accuracy of 10^{-15}, i.e., three additional iterations in comparison to the CCR strategy. Two local V-cycles improve the situation such that the delay in finding the solution is reduced to one iteration. A local correction by two W-cycles or F-cycles deliver almost the same residual error and reduces the delay in comparison to the two local V-cycles strategy. A F-cycle is preferred, since its computational cost is by a factor of $8/7$ lower than for a W-cycle. Further in Fig. 8, we vary the number of V-cycles k_F for solving the subproblem on Ω_F. The delay in finding the solution significantly depends on how accurate the subproblem is solved. One local V-cycle reduces the solution process by one iteration in comparison to no-local-recovery strategy, two

DOI: 10.1002/nla

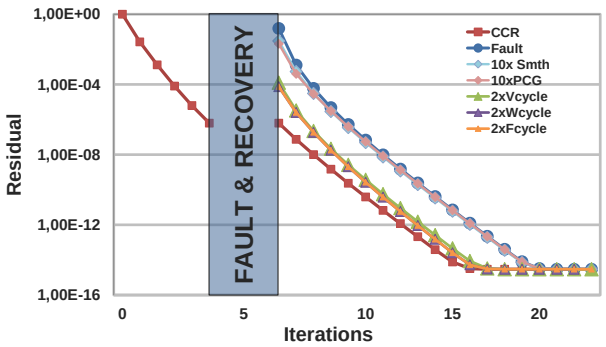

Fig. 7. Convergence of the residual error scaled by the initial residual for different local recovery strategies.

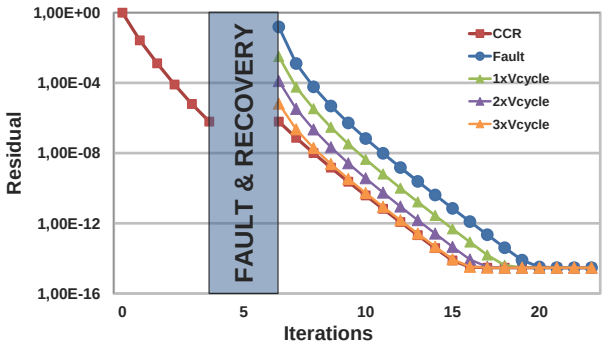

Fig. 8. Convergence of the residual error scaled by the initial residual for different number k_F of local V-cycles.

V-cycles by three iterations and three V-cycles completely compensate the fault. Additionally, we compare the size of κ for different local recovery strategies and for different iterations after which a failure occurs. We study faults after 5, 7, 11 iterations, respectively, and smoothing, PCG-iterations, and multigrid V-, W-, and F-cycle as local recovery strategies. The V-cycle convergence rate has been numerically evaluated as $\mu = 0.194$. The cycle advantage κ is measured after 15 iterations ($K = 15$) and presented in Tab. I.

The residual error of the CCR strategy is of order 10^{-14} after 15 iterations such that the convergence does not saturate due to round off errors. We need 4.164, 6.445 or 10.999 additional V-cycles for the case of no-local-recovery and when the failure occurs after 5, 7 or 11 iterations, respectively, in order to achieve the same accuracy as in the CCR strategy. This is due to the error difference, introduced by the fault, before and after the fault which is larger the later it occurs. These are also the maximal κ values which can be achieved by a local recovery strategy. For the fault after 5 iterations in Tab. I, we obtain similar κ for a local recovery with 4 V-cycles, 3 W-cycles, or 3 F-cycle which is almost as good as in the CCR strategy. Further, for smoothing and PCG iterations, only marginally small improvement can be observed in comparison to using multigrid cycles or the CCR strategy. As expected for the fault after 7 iterations, 5 W- or F-cycles and for the fault after 11 iterations, more than 5 W- or F-cycles are necessary to achieve a κ similar to the CCR strategy. For the other strategies (smoothing and PCG iterations) no significant improvement is obtained. Again, we observe that W-cycles and F-cycles yield similar results, thus, F-cycles are preferred to W-cycles due to computational cost arguments.

DOI: 10.1002/nla

Tab. I. Cycle Advantage for an early (after 5 iterations) , middle (after 7 iterations) , and late (after 11 iterations) fault for different local recovery strategies.

Fault After 5 Iter.		**Fault After 7 Iter.**		**Fault After 11 Iter.**	
Strategies	κ	Strategies	κ	Strategies	κ
CCR	4.164	CCR	6.445	CCR	10.999
1 × Vcycle	1.637	1 × Vcycle	1.646	1 × Vcycle	1.678
2 × Vcycle	3.075	2 × Vcycle	3.111	2 × Vcycle	3.225
3 × Vcycle	4.069	3 × Vcycle	4.445	3 × Vcycle	4.643
4 × Vcycle	4.165	4 × Vcycle	5.693	4 × Vcycle	5.970
5 × Vcycle	4.165	5 × Vcycle	6.397	5 × Vcycle	7.242
1 × Wcycle	1.825	1 × Wcycle	1.780	1 × Wcycle	1.747
2 × Wcycle	3.425	2 × Wcycle	3.404	2 × Wcycle	3.380
3 × Wcycle	4.147	3 × Wcycle	4.967	3 × Wcycle	4.951
4 × Wcycle	4.164	4 × Wcycle	6.249	4 × Wcycle	6.458
5 × Wcycle	4.165	5 × Wcycle	6.444	5 × Wcycle	7.911
1 × Fcycle	1.828	1 × Fcycle	1.781	1 × Fcycle	1.748
2 × Fcycle	3.426	2 × Fcycle	3.405	2 × Fcycle	3.381
3 × Fcycle	4.147	3 × Fcycle	4.967	3 × Fcycle	4.951
4 × Fcycle	4.164	4 × Fcycle	6.249	4 × Fcycle	6.458
5 × Fcycle	4.165	5 × Fcycle	6.444	5 × Fcycle	7.911
2 × PCG	0.007	2 × PCG	0.012	2 × PCG	0.039
5 × PCG	0.026	5 × PCG	0.043	5 × PCG	0.104
10 × PCG	0.037	10 × PCG	0.072	10 × PCG	0.183
10 × Smth	0.105	10 × Smth	0.147	10 × Smth	0.249
20 × Smth	0.172	20 × Smth	0.236	20 × Smth	0.387
30 × Smth	0.221	30 × Smth	0.298	30 × Smth	0.482
40 × Smth	0.260	40 × Smth	0.346	40 × Smth	0.554
50 × Smth	0.291	50 × Smth	0.384	50 × Smth	0.611

4. CONCLUSION AND OUTLOOK

This paper gives a first insight in constructing a fault tolerant multigrid solver. It is shown that geometric multigrid solvers are inherently suitable to deal with failures of processes. The failure results in a loss of the values of a subdomain. To recover these lost values local subproblems with Dirichlet boundary conditions are solved by various strategies ranging from relaxation scheme, Krylov space methods to multigrid cycles. Further, the local problems are accelerated by a *superman* strategy through additional parallelization of the recovery. We introduce the reference parameter *Cycle Advantage* which gives the possibility to demonstrate to which extent a recovery strategy improves the time-to-solution in terms of cycle applications in comparison to use no-recovery. Only multigrid cycles can efficiently treat the local subproblem and even reach the same accuracy as the complete check-pointing recovery but with a minimized access to backup memory.

In future work, we will extend this basic idea for problems with more unknowns, incorporate the local strategy in a global recovery strategy to balance the waiting time for all processors. We will enhance our considerations to problems with non-vanishing right-hand side and analyze its recovery. We will further develop acceleration strategies for the local recovery and include them in the HHG framework with a fault tolerant MPI-variants.

DOI: 10.1002/nla

REFERENCES

1. Cappello F. Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities. *International Journal of High Performance Computing Applications* 2009; **23**(3):212–226.
2. Bergen BK, Hülsemann F. Hierarchical hybrid grids: Data structures and core algorithms for multigrid. *Numerical Linear Algebra with Applications* 2004; **11**(2-3):279–291.
3. Gmeiner B, Rüde U, Stengel H, Waluga C, Wohlmuth B. Performance and scalability of hierarchical hybrid multigrid solvers for Stokes systems. *SIAM Journal on Scientific Computing* 2014; Accepted.
4. Gmeiner B, Köstler H, Stürmer M, Rüde U. Parallel multigrid on hierarchical hybrid grids: a performance study on current high performance computing clusters. *Concurrency and Computation: Practice and Experience* 2014; **26**(1):217–240.
5. Sundar H, Biros G, Burstedde C, Rudi J, Ghattas O, Stadler G. Parallel geometric-algebraic multigrid on unstructured forests of octrees. *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, 2012; 1–11.
6. Maniatakos M, Kudva P, Fleischer B, Makris Y. Low-cost concurrent error detection for floating-point unit (FPU) controllers. *IEEE Transactions on Computers* July 2013; **62**(7):1376–1388.
7. Düben PD, Joven J, Lingamneni A, McNamara H, De Micheli G, Palem KV, Palmer TN. On the use of inexact, pruned hardware in atmospheric modelling. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 2014; **372**(2018).
8. Mukherjee SS, Emer J, Reinhardt SK. The soft error problem: An architectural perspective. *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, IEEE, 2005; 243–247.
9. Malkowski K, Raghavan P, Kandemir M. Analyzing the soft error resilience of linear solvers on multicore multiprocessors. *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, 2010; 1–12.
10. Bosilca G, Bouteiller A, Brunet E, Cappello F, Dongarra J, Guermouche A, Herault T, Robert Y, Vivien F, Zaidouni D. Unified model for assessing checkpointing protocols at extreme-scale. *Concurrency and Computation: Practice and Experience* 2014; **26**(17):2772–2791.
11. Zheng G, Huang C, Kalé LV. Performance evaluation of automatic checkpoint-based fault tolerance for ampi and charm++. *ACM SIGOPS Operating Systems Review* 2006; **40**(2):90–99.
12. Di S, Bouguerra MS, Bautista-Gomez L, Cappello F. Optimization of multi-level checkpoint model for large scale HPC applications. *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*, IPDPS '14, IEEE Computer Society: Washington, DC, USA, 2014; 1181–1190.
13. Bland W, Du P, Bouteiller A, Herault T, Bosilca G, Dongarra JJ. Extending the scope of the checkpoint-on-failure protocol for forward recovery in standard mpi. *Concurrency and Computation: Practice and Experience* 2013; **25**(17):2381–2393.
14. Huang KH, Abraham JA. Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on Computers* Jun 1984; **33**(6):518–528.
15. Bridges PG, Ferreira KB, Heroux MA, Hoemmen M. Fault-tolerant linear solvers via selective reliability. *ArXiv e-prints* Jun 2012; .
16. Casas M, de Supinski BR, Bronevetsky G, Schulz M. Fault resilience of the algebraic multi-grid solver. *Proceedings of the 26th ACM International Conference on Supercomputing*, ICS '12, ACM: New York, NY, USA, 2012; 91–100.
17. Cappello F, Geist A, Gropp B, Kale L, Kramer B, Snir M. Toward exascale resilience. *International Journal of High Performance Computing Applications* Nov 2009; **23**(4):374–388.
18. Cappello F, Geist A, Kale S, Kramer B, Snir M. Toward exascale resilience: 2014 update. *Supercomputing frontiers and innovations* 2014; **1**:1–28.
19. Anfinson J, Luk FT. A linear algebraic model of algorithm-based fault tolerance. *IEEE Transactions on Computers* 1988; **37**(12):1599–1604.
20. Boley DL, Brent RP, Golub GH, Luk FT. Algorithmic fault tolerance using the Lanczos method. *SIAM Journal on Matrix Analysis and Applications* 1992; **13**(1):312–332.
21. Chen Z, Dongarra J. Algorithm-based fault tolerance for fail-stop failures. *IEEE Transactions on Parallel and Distributed Systems* 2008; **19**(12):1628–1641.
22. Luk FT, Park H. An analysis of algorithm-based fault tolerance techniques. *Journal of Parallel and Distributed Computing* Apr 1988; **5**(2):172–184.
23. Davies T, Chen Z. Correcting soft errors online in LU factorization. *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '13, ACM: New York, NY, USA, 2013; 167–178.
24. Sloan J, Kumar R, Bronevetsky G. Algorithmic approaches to low overhead fault detection for sparse linear algebra. *Proceedings of the 2012 42Nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, DSN '12, IEEE Computer Society: Washington, DC, USA, 2012; 1–12.
25. Chen Z. Online-ABFT: An online algorithm based fault tolerance scheme for soft error detection in iterative methods. *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '13, ACM: New York, NY, USA, 2013; 167–176.
26. Roy-Chowdhury A, Banerjee P. A fault-tolerant parallel algorithm for iterative solution of the laplace equation. *Parallel Processing, 1993. ICPP 1993. International Conference on*, vol. 3, 1993; 133–140.
27. Stoyanov MK, Webster CG. Numerical analysis of fixed point algorithms in the presence of hardware faults. *Technical Report*, Tech. rep. Oak Ridge National Laboratory (ORNL) Aug 2013.
28. Cui T, Xu J, Zhang CS. An Error-Resilient Redundant Subspace Correction Method. *ArXiv e-prints* Sep 2013; .
29. Brandt A, Livne OE. *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics, Revised Edition*. Classics in Applied Mathematics, Society for Industrial and Applied Mathematics, 2011.
30. Hackbusch W. *Multi-grid methods and applications*, vol. 4. Springer-Verlag Berlin, 1985.
31. Harding B, Hegland M, Larson J, Southern J. Scalable and fault tolerant computation with the sparse grid combination technique. *ArXiv e-prints* Apr 2014; .

DOI: 10.1002/nla