University of Stuttgart
Department of Mathematics


# Fault-tolerant numerical methods

## Milestone Presentation

October 9, 2018

Mirco Altenbernd

ians

SimTech

# Motivation - Fault-tolerance

- More components at exascale $\Rightarrow$ higher probability of faults/failures

| #cores | 1 | 100 | 10 000 | 1 000 000 |
|---|---|---|---|---|
| **MTBF** | 5 years | 18 days | 4 hours | 3 mins |

- Active debates to sacrifice reliability for energy efficiency
- Nightmare scenarios of MTBF $< 1$ h

- Two main types of faults:
  - *Hard faults*
    Interrupt the user's program
  - *Soft faults*
    Do not *immediately* interrupt the user's program
- Often further classified: *Transient*, *sticky*, *persistent*, *silent*, ...

J. Elliot, M. Hoemmen, F. Mueller, **Evaluating the Impact of SDC on the GMRES Iterative Solver**, IEEE 28th International Parallel and Distributed Processing Symposium, 2014

# Motivation - Fault-tolerance

## Failures

- A fault becomes a failure if it impacts the user, e.g.:
  Hard faults result in failures if the user is running an application
- If a soft fault leads to an incorrect solution it becomes a *silent failure*

## Focus of our research

- Faults introduced by *silent data corruption* (SDC): Stored data is not changed but the result of a computation, e.g., for $a = b = 2$ we receive $c = a + b = 5$
- Node-losses which are a variation of hard faults/failures

# Motivation - Fault-tolerance

## Classical techniques

- Reliability in hardware (ECC protection etc.) too power-hungry
- Checkpoint-restart too memory-intensive (and too slow)
- Triple modular redundancy too power-hungry, but: can be more energy-efficient and applicable for large fault rates

## Algorithm-based fault-tolerance

- Exploit algorithmic properties to detect and correct faults on-the-fly
- Can be more efficient than middleware-based obvious solutions
- Often provable error bounds

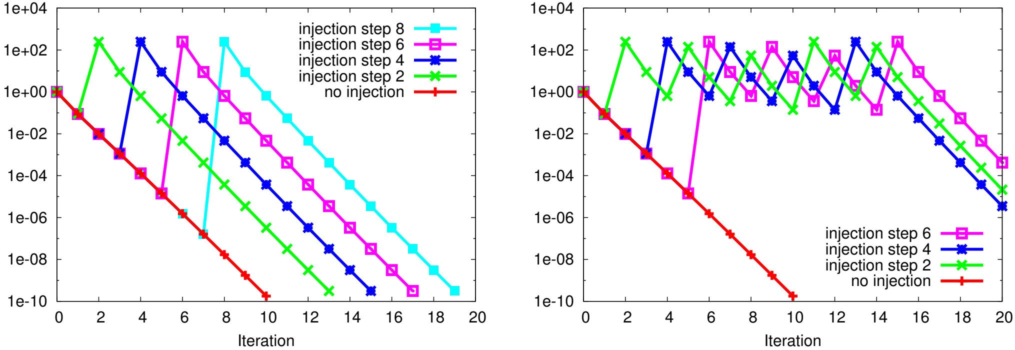
# Algorithm-based fault-tolerance

## Challenges

- Requires custom modifications for each method
- Overhead in the fault-free scenario should be small
- False-positives should be rare without much impact on convergence
- MPI: Faults can result in node-losses
  $\Rightarrow$ Actually a program can not react on a crashed MPI rank
- Provability behind heuristics

Initial focus on multigrid because of its behaviour in presence of faults.

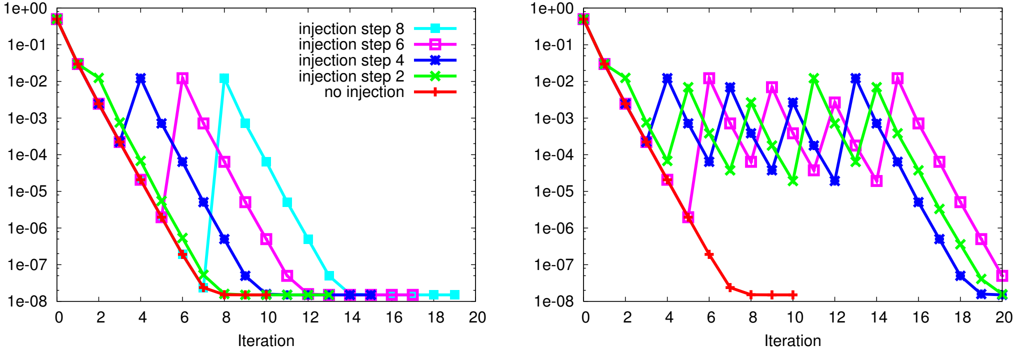
# Multigrid and faults




## Observations

- A fault is comparable to a restart of the multigrid solver
- Multigrid converges always if the fault-rate is not to high
- Note: SDC and node-losses results in similar behaviour

# Multigrid and faults




## Observations

- A fault is comparable to a restart of the multigrid solver
- Multigrid converges always if the fault-rate is not to high
- Note: SDC and node-losses results in similar behaviour

# Multigrid and faults

Assumption: Multigrid is self-stabilising

**Self-stabilising**

Starting from any state the solver comes back to a valid state.

P. Sao, R. Vuduc, **Self-stabilizing Iterative Solvers**, 2013

- Original defined by Dijkstra in 1974 for systems of distributed control
- Examples: Newton- and Jacobi-methods

# Multigrid and faults

Assumption: Multigrid is self-stabilising

## Sketch of the proof

- Multigrid is a defect correction procedure, i.e., a fixed point iteration
- Hackbusch's multigrid convergence proof is based on contraction arguments:

If the contraction property holds for a given iteration operator, then convergence of the corresponding iteration procedure is guaranteed for any initial guess

- Basically Banach's fixed point theorem
- The new initial guess is simply the last iterate with some faulty entries
- Matrices and grid transfer operators are fault-free
  $\Rightarrow$ Contraction property is not affected

# Ongoing subprojects

❶ Compressed checkpointing

- Using compression techniques to decrease checkpoint size
- Locally restore lost or faulty data from compressed checkpoint
- Improve restoration by solving local auxiliary problems

❷ SDC-tolerant multigrid

- Increase the inherent robustness of multigrid with respect to bit-flips
- Apply a local smoothing stage protection to detect and repair soft faults

❸ User level exception handling

- User-friendly asynchronous C++ MPI interface for parallel exception handling
- Propagate exceptions with MPI to always ensure same state on all ranks
- Developed for MPI-4 with an MPI-3 fall-back

# Compressed checkpointing

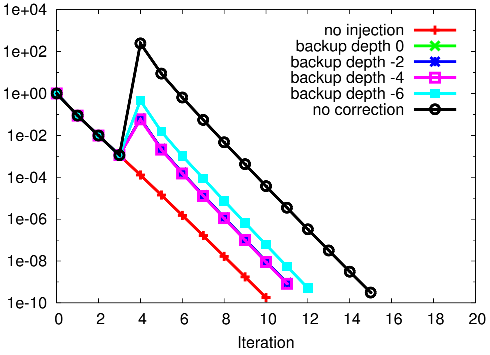
**Research goal**

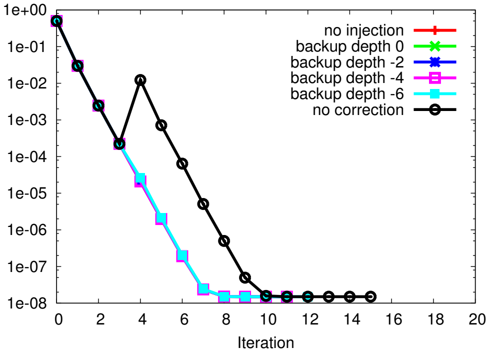
Reduce size of checkpoints and restore lost data efficiently

# ❶ Compressed checkpointing

## Multigrid compression

- Use multigrid transfer operators to compress checkpoint
- Data reduction in $d$ dimensions: $\sim 2^d$ per level (backup depth)
- Restore lost data with prolongated checkpoint

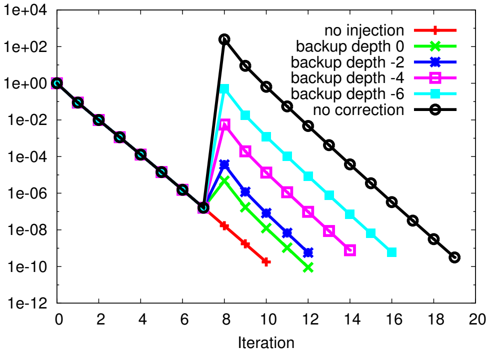


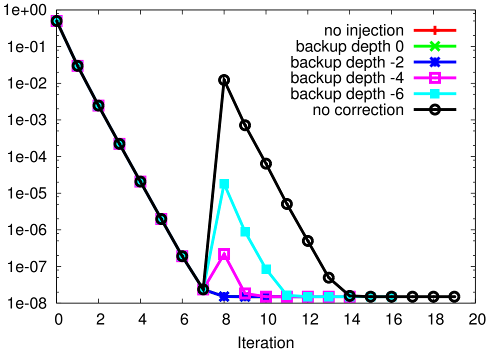



# ❶ Compressed checkpointing

## Multigrid compression

- Use multigrid transfer operators to compress checkpoint
- Data reduction in $d$ dimensions: $\sim 2^d$ per level (backup depth)
- Restore lost data with prolongated checkpoint

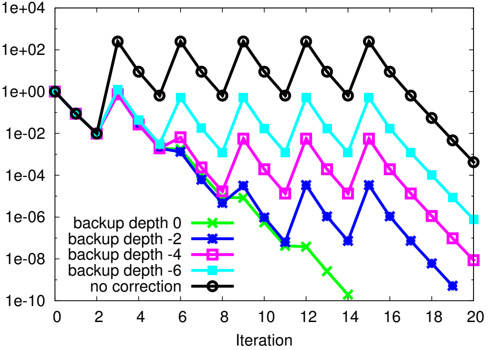


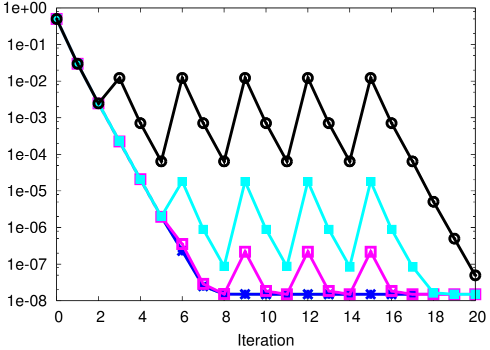



# ❶ Compressed checkpointing

## Multigrid compression

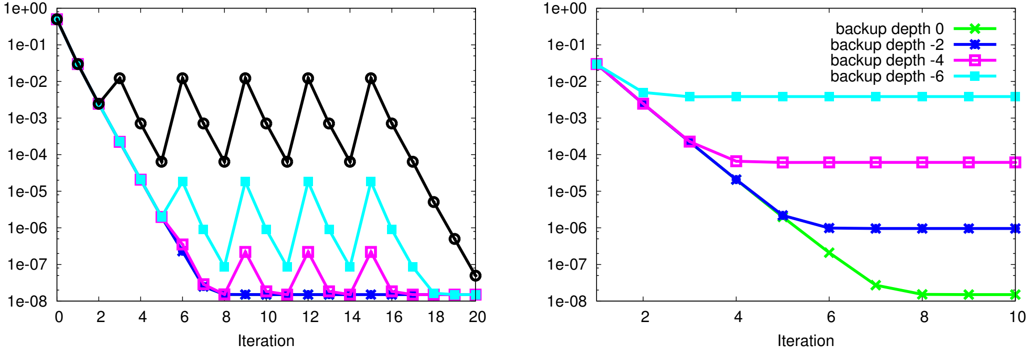
- Use multigrid transfer operators to compress checkpoint
- Data reduction in $d$ dimensions: $\sim 2^d$ per level (backup depth)
- Restore lost data with prolongated checkpoint



# ❶ Compressed checkpointing

## Limits of multigrid compression




- Discretisation error dominates at some point
- Dominates earlier for highly compressed data
- Factor between $L^2$-quality and $L^2$-error depends on amount of repaired data

# ❶ Compressed checkpointing

## Problem

- Convergence for late faults can not be restored with highly compressed backups
- Recurrent faults need even less compressed checkpoints

## Solution

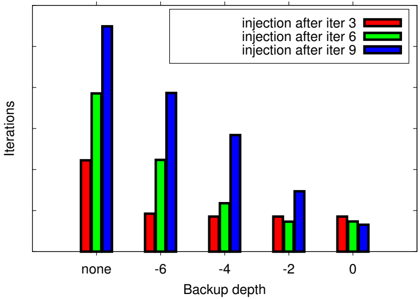
- Solve an auxiliary problem with Dirichlet boundary to improve accuracy
- Use decompressed data as initial guess

## Auxiliary problem

Extend faulty indices $\mathcal{F} \subset \mathbb{N}$ by connectivity pattern of Operator $\mathbf{A}$ to $\mathcal{J}$ and solve

$$\begin{aligned} \mathbf{A}(\mathcal{J},\mathcal{J})\tilde{x}(\mathcal{J}) &= b(\mathcal{J}) && \text{in } \mathcal{F} \\ \tilde{x} &= x && \text{on } \mathcal{J}\backslash\mathcal{F} \end{aligned}$$

iteratively with initial guess $\tilde{x} = x_{cp}$ in $\mathcal{F}$.



# ❶ Compressed checkpointing

**Summary:** Multigrid compression

- Multigrid compressed checkpoints can be used to recover from faults
- Early on highly compressed data is sufficient
- Later compression rate has to be decreased
- Eventually an auxiliary problem has to be solved to ensure convergence
- The decompressed data is a good initial guess for this auxiliary problem
- Same idea could be used with other hierarchic methods

**But**

Multigrid is good preconditioner, but rarely a standalone solver

D. Göddeke, M.A., D. Ribbrock, **Fault-tolerant finite-element multigrid algorithms with hierarchically compressed asynchronous checkpointing**, Parallel Computing, 2015

# ❶ Compressed checkpointing

## Extend the idea to other solvers and methods

- Restore lost/faulty data from different kind of checkpoints:
  - Checkpoints based on different compression techniques:
    - Lossy compression: Multigrid compression, SZ compression, . . .
    - Lossless compression: *zip*, *png*, *gif*, . . .
  - Checkpoints which are less frequent, i.e. from previous iterations
  - Replace with a full roll-back instead of partial replacement
  - . . . and combinations

**Task**

Compare quality and efficiency of repair with different combinations

# ❶ Compressed checkpointing

## SZ compression

- Save initial point value (with reduced accuracy) as unpredictable data
- Predict next point based on previous processed points via an interpolation based on $n$ layers
- Compare predicted value with real value and improve it through a *Huffman*-like coding
- If still not ‘close enough’ data is stored as unpredictable and compressed via binary-representation analysis

## Advantages and disadvantages

- Adaptive controllable compression rate (via `error_bound`)
- More computational overhead than inherent multigrid compression
- No random-access to single decompressed values

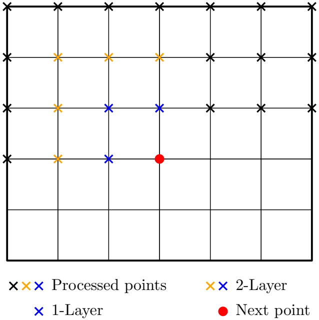
D. Tao, S. Di, Z. Chen and F. Cappello, **Significantly Improving Lossy Compression for Scientific Data Sets Based on Multidimensional Prediction and Error-Controlled Quantization**, Computing Research Repository, 2017

# ❶ Compressed checkpointing

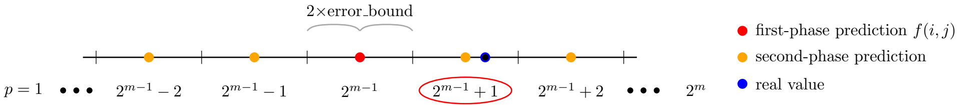
## SZ compression (version 1.4.2, 2D)

- Predict values row by row (top to bottom, left to right)
- $\mathcal{V} = \{V(i,j)\}$: set of already compressed point values
- Interpolation based first-phase prediction $f(i,j)$

| | |
|---|---|
| 1-Layer | $V(i,j-1) + V(i-1,j) - V(i-1,j-1)$ |
| 2-Layer | $2V(i,j-1) + 2V(i-1,j) - 4V(i-1,j-1)$ $-V(i,j-2) - V(i-2,j) + 2V(i-2,j-1)$ $+2V(i-1,j-2) - V(i-2,j-2)$ |
| ... | ... |




- $2^m$ intervals with size of $2\times$`error_bound` around $f(i,j)$




- Store index $p$ or and $p = 0$ and compressed binary-representation if the real value is not in any second-phase prediction interval
- Data is decompressed via interpolation and shifted by the Huffman-code

# ❶ Compressed checkpointing

## Numerical tests

- Rotated anisotropic diffusion in 2D

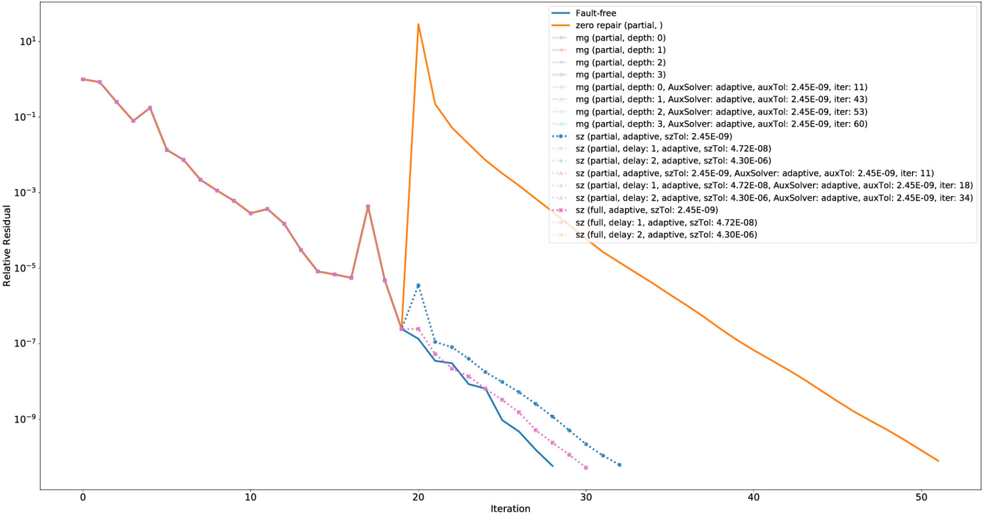
$$-\nabla \cdot (\mathbf{Q}\mathbf{A}\mathbf{Q}^T \nabla u) = b$$

  with rotation matrix $\mathbf{Q}$ and diffusion matrix $\mathbf{A} = \begin{pmatrix} 1 & 0 \\ 0 & \varepsilon \end{pmatrix}$.
- Linear finite elements on rectangular grid with $200 \times 300$ degrees of freedom
- Solver: BiCGStab
- Preconditioner: Algebraic multigrid (one V-cycle, smoothed aggregation)

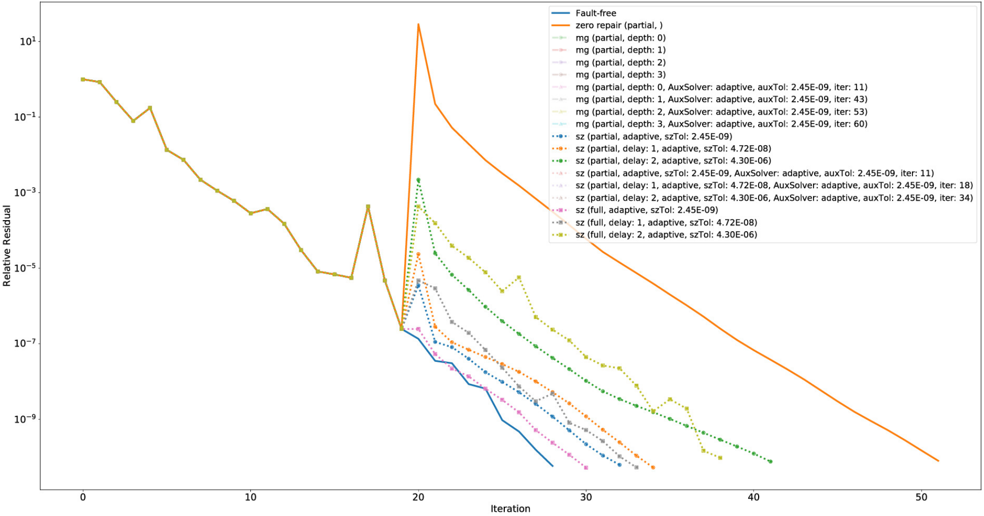
**Challenge**

Recover from a fault in the outer solver with a compressed checkpoint

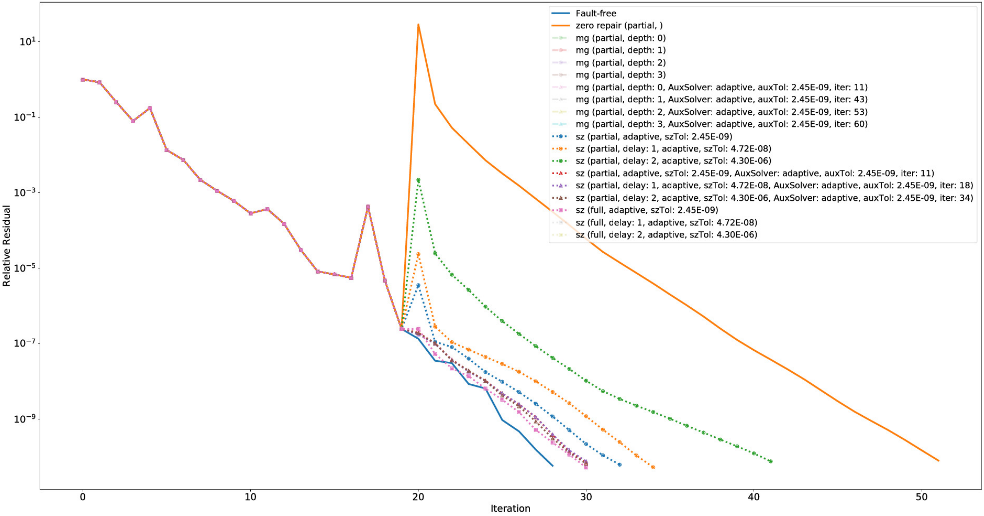
# ❶ Compressed checkpointing



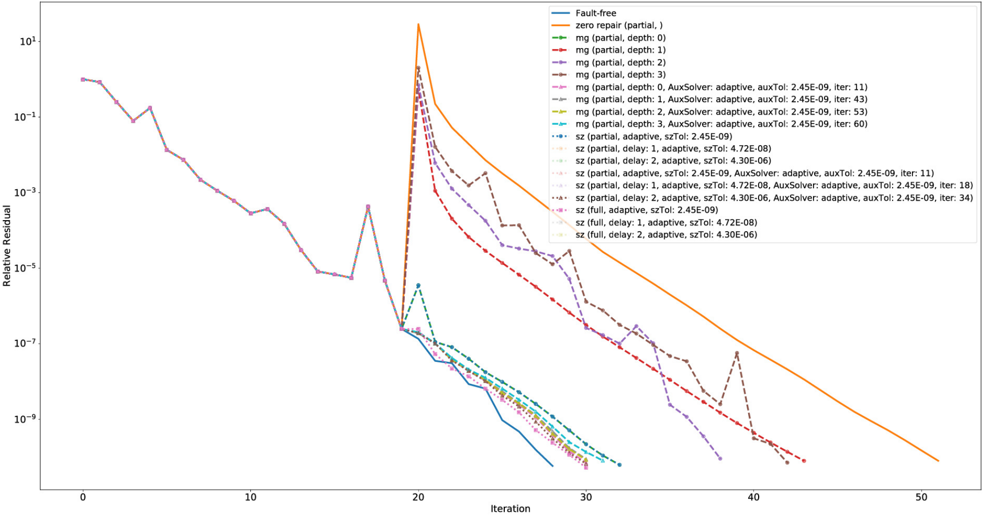
# ❶ Compressed checkpointing



# ❶ Compressed checkpointing



# ❶ Compressed checkpointing

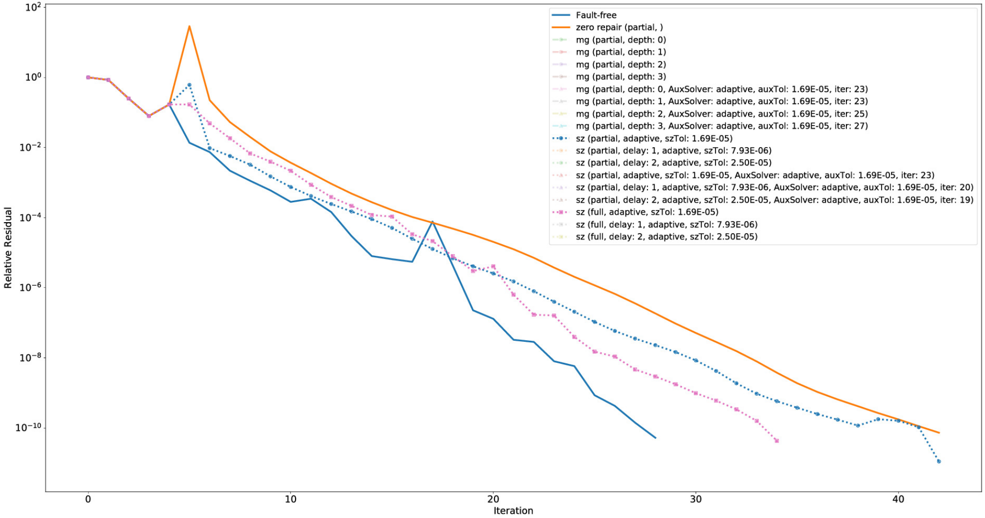


# ❶ Compressed checkpointing

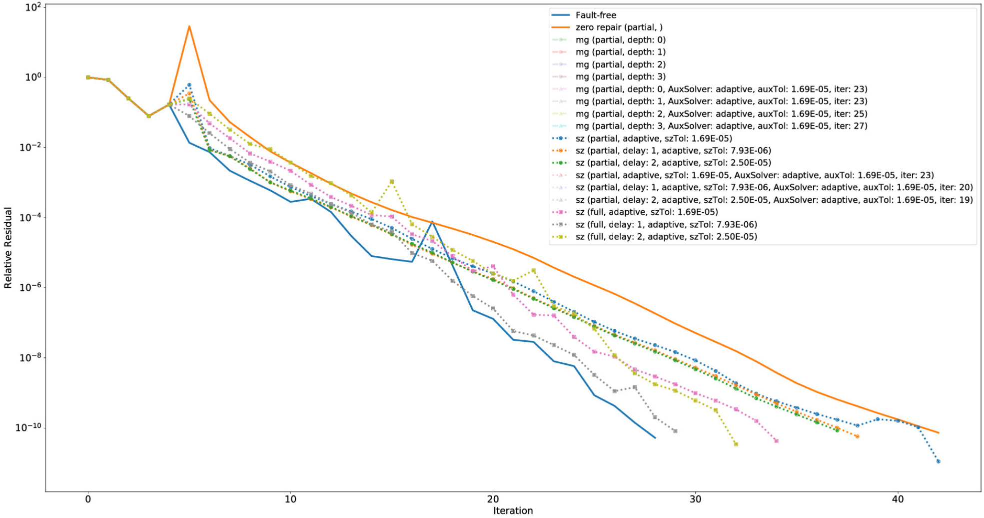
**Summary** (late fault)

- SZ compression with a high accuracy seems promising for repair
- Full repair is better than partial repair
- A delay significantly deteriorates the quality of repair
- Auxiliary solver improves quality to the level of SZ compression without delay
- Multigrid compression with auxiliary solver works similar
- Greater delay/depth increases iteration count of auxiliary solver

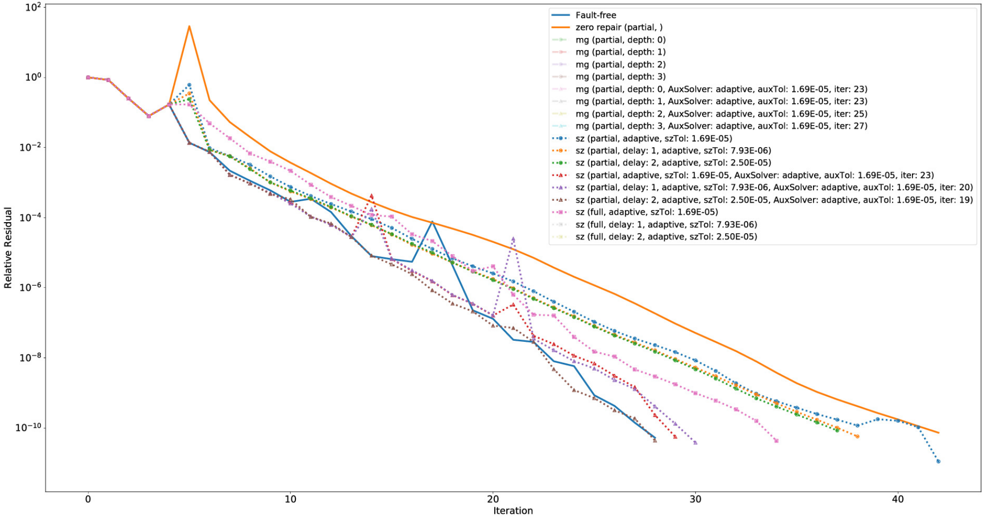
# ❶ Compressed checkpointing



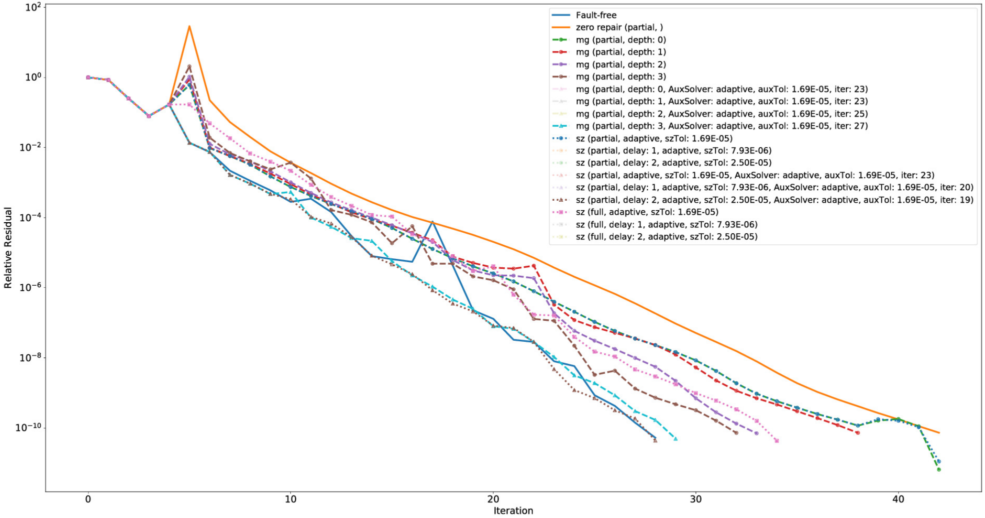
# ❶ Compressed checkpointing



# ❶ Compressed checkpointing



# ❶ Compressed checkpointing



# ❶ Compressed checkpointing

**Summary** (early fault)

- Despite increased *tolerance scaling* full and partial repair with a SZ compressed backup is not good
- An auxiliary solver can still restore good convergence
- Even strong multigrid compression works better than SZ compression without an auxiliary solver
- Multigrid with auxiliary solver also restores convergence with a similar amount of iterations

**Open task**

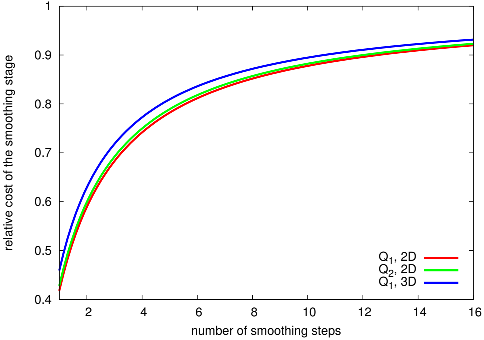
Develop a performance model to find the most effective combination

# SDC-tolerant multigrid

**Research goal**

Increase the robustness of multigrid with respect to silent data corruption

# ❷ SDC-tolerant multigrid




## Observation

Most time is spent within the smoothing stage

## Idea

- Don't ensure correctness value by value
- Only verify if output of the smoothing stage is 'good enough'
- Use invariants of *Full Approximation Scheme multigrid* (FASMG) to test output
- Protect remaining part (transfer phase and coarse grid correction) with checksums

# ❷ SDC-tolerant multigrid

## Differences between MG and FASMG

- Multigrid's correction problem is given by

$$\mathbf{A}_k(u_k + v_k) = b_k$$

- Classic MG uses linearity and searches on the next coarser level for the correction $v_k$ only

$$\mathbf{A}_{k-1}v_{k-1} = \mathbf{R}^k_{k-1}(b_k - \mathbf{A}_k u_k)$$

- FASMG searches always for the full solution $\tilde{v}_k := u_k + v_k$

$$\mathbf{A}_{k-1}\tilde{v}_{k-1} = \mathbf{R}^k_{k-1}(b_k - \mathbf{A}_k u_k) + \mathbf{A}_{k-1}\mathbf{I}^k_{k-1}u_k$$

- $\tilde{v}_{k-1}$ is an approximation to the fine grid problem but with lower resolution
- We can interpret $\tilde{v}_{k-1}$ as a compressed backup of $\tilde{v}_k$
- $\mathbf{R}^k_{k-1}$ and $\mathbf{I}^k_{k-1}$ are different restriction operators

# ❷ SDC-tolerant multigrid

## STMG algorithm

**Call** : $\texttt{STMG}(k, \mathcal{A}, b, u^{(0)})$

$u^{(\nu)} = \mathcal{S}^{\nu}\left(u^{(0)}, b\right)$ // pre-smoothing

$r_k = b - \mathbf{A}_k u^{(\nu)}$

$\texttt{check\_and\_repair\_res}(r_k, k)$

$r_{k-1} = \mathbf{R}^{k}_{k-1} r_k$

$\tilde{v}^{(0)}_{k-1} = \mathbf{I}^{k}_{k-1} u^{(\nu)}$

$r_{k-1} = r_{k-1} + \mathbf{A}_{k-1} \tilde{v}^{(0)}_{k-1}$

$\tilde{v}_{k-1} = \texttt{STMG}(k-1, \mathcal{A}, r_{k-1}, \tilde{v}^{(0)}_{k-1})$ // coarse grid correction

$c = \tilde{v}_{k-1} - \tilde{v}^{(0)}_{k-1}$

$\texttt{check\_and\_repair\_cor}(c, k-1)$

$\tilde{u}^{(\nu)} = u^{(\nu)} + \mathbf{P}^{k-1}_{k}(c)$

$u = \mathcal{S}^{\mu}\left(\tilde{u}^{(\nu)}, b\right)$ // post-smoothing

**if** *on fine grid* **then**

  $\texttt{check\_and\_repair\_res}(b - \mathbf{A}_L u, k)$

**end**

- $k$ denotes the current grid level
- $\mathbf{P}^{k-1}_{k}$ is the prolongation operator
- $\mathcal{S}^{\nu}$ is the smoother which is applied $\nu$ times
- Direct solver on coarsest grid

# ❷ SDC-tolerant multigrid

## Check and repair algorithm (correction)

- Check output of smoother through element-wise comparison
- Threshold based on residual/correction norm (scaled by tolerance factor): Converges monotonously to zero if operator is s.p.d.
- Transfer (scale) to next level grid with transfer operator norm
- Store ‘faulty’ indices in set $\mathcal{L}$ and repair:

**if** $\mathcal{L} \neq \emptyset$ **then**
  $\tilde{v}_{k-1} = \mathbf{I}_{k-1}^{k} \tilde{v}_{k}^{(0)}$ `// restrict (non-faulty) initial approximation`
  $\tilde{c} = u_{k-1} - \tilde{v}_{k-1}$ `// calculate new correction vector`
  $\tilde{c} = \mathbf{P}_{k}^{k-1} \tilde{c}$ `// prolongate new correction vector`
  **for** $i \in \mathcal{L}$ **do**
    $c(i) = \tilde{c}(i)$ `// replace faulty components`
  **end**
**end**

- Residual check and repair works similar but easier
- Assumption: Coarse grid solver output is fault-free

# ❷ SDC-tolerant multigrid

## Numerical tests

- V-cycle multigrid with $4 + 4$ Jacobi smoothing steps
- $1$ million degrees of freedom, $Q_1$ Lagrange Finite Elements
- $4000$ different fault scenarios per test problem
- Fault probability of $10^{-7}$ per degree of freedom
  $\Rightarrow$ Approximately once every 10th smoothing step on fine grid
  $\Rightarrow$ Approximately twice every multigrid iteration

| | **diff** | **diff-conv** | **andiff** | **andiff-conv-reac** |
|---|---|---|---|---|
| **fault-free** | 4 | 6 | 14 | 7 |
| **MG** (div.) | 4.225 (6.8%) | 6.268 (8.4%) | 15.111 (21.3%) | 7.466 (11%) |
| **STMG** | 4.038 | 6.007 | 14.007 | 7.017 |

- Nearly no false-positives: Approximately $15$ in $4000$ runs

M.A. and D. Göddeke, **Soft fault detection and correction for multigrid**, International Journal of High Performance Computing Applications, 2017

# ❷ SDC-tolerant multigrid

## Numerical overhead

- Overhead of FASMG is approximately $20\%$
- Smoother protection itself results in an overhead of $4\%$
- Checksums lead to additional $5\%$ ($8\times$ Jacobi smoothing)

| | unprotected (MG) | unprotected (FASMG) | defect correction (checksums) | smoothing stage (new algorithm) | STMG (both) |
|---|---|---|---|---|---|
| **time** | 35.49 | 43.02 | 45.23 | 44.76 | 46.18 |
| **factor** | 0.825 | 1 | 1.051 | 1.040 | 1.073 |
| **factor** | 1 | 1.212 | 1.274 | 1.261 | 1.301 |

$\Rightarrow$ Overall overhead of less than $30\%$ compared to classical MG

# ❷ SDC-tolerant multigrid

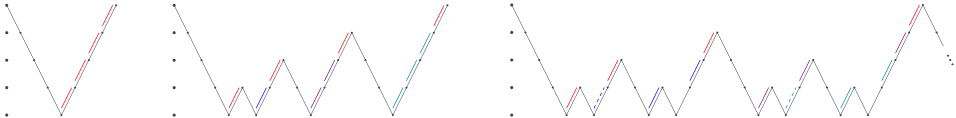
## Applicability

- Geometric and algebraic multigrid (AMG)
- Standalone and as preconditioner
- Serial and parallel:

| #it | 17 | 18 | 19 | 20 | 21 | 25 | 34 | 41 | div | avg |
|---|---|---|---|---|---|---|---|---|---|---|
| **AMG** | 97 | 1 | | | 2 | 1 | 2 | 1 | 87 | 17.72 |
| **STAMG** | 179 | 4 | 6 | 2 | | | | | 0 | 17.12 |

Parallel execution of protected algorithm on 4 procs with CG and AMG preconditioner.

- All cycle types:



Visualisation of V-, F- and W-cycle multigrid.

# User level exception handling

**Research goal**

Extend the functionality of MPI for fault-tolerant algorithms

# ❸ User level exception handling

## Challenges

- Detect locally thrown exceptions
- Inform all processes of the error
- Wrap it into a user-friendly `C++` compliant interface
- Support asynchronous communication (similar to `C++` future concept)

## Code Example

```
try{ // scope to be protected
  Guard guard(communicator);
  Future f = init_communication();
  do_some_computation();
  f.get(); // MPI::Wait()
}catch(...) {
  // handle thrown exceptions
}
```

- Cheap guard object protects *try* block
- Is destructed during stack unwinding
- Propagate exception across communicator (uses `std::uncaught_exception`)

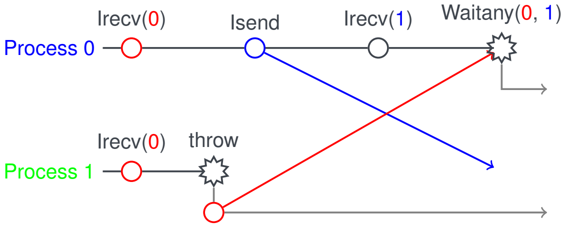
# ❸ User level exception handling

## MPI-4 variant

- Interface is using the User-level failure-mitigation extension (ULFM)
- Provides functionality for
  - Hard fault *detection*
  - Communicator *revocation*
  - *Shrinking* of faulty communicator (i.e. excluding faulty processors)

## MPI-3 variant

- Fall-back library which creates additional communication channel for exceptions
- Drawback: cannot interrupt MPI collectives, no hard fault protection




- `https://gitlab.dune-project.org/exadune/blackchannel-ulfm`

C. Engwer, M. A., N. Dreier, D. Göddeke, **A High-Level C++ Approach to Manage Local Errors, Asynchrony and Faults in an MPI Application**, Proceedings of PDP 2018, 2018

# Summary and Outlook

# Summary

- We developed three ‘orthogonal’ approaches to increase fault-tolerance, especially for multigrid algorithms:
  - Efficient SDC protection with build in properties
  - Restoration with compressed checkpoints:
    MG and SZ compression both have their (dis-)advantages
  - Exception-propagation to ensure same state in MPI programs
- ’User level exception handling’ can be used for many algorithms to develop MPI-4 ready fault-tolerant algorithms already in MPI-3
- Interface is ready for asynchronous algorithms (*future* concept):
  - Asynchronous checkpointing/repair
  - Local-failure local-recovery
  - …

# What’s next?

- Integrating the new MPI interface into DUNE[1]
- Improving features/functionality of the interface for wider applicability
- Evaluating and combining developed concepts:
  - Switch between different compression and repair techniques:
    Adaptively select the most efficient one
  - Asynchronous checkpointing/repair
  - Asynchrony in multigrid:
    Local smoothing while restoring lost processors?
  - ...

Thinking about **ideas for fault-tolerance and asynchrony in remaining PDE solver parts**, not only linear solver

[1] funded by DFG: German Priority Programme 1648, SPPEXA, EXADUNE

# Acknowledgements


- Dominik Göddeke (University of Stuttgart)
- Nils-Arne Dreier and Christian Engwer (University of Münster)
- Jon Calhoun (Clemson University, South Carolina, USA)

- DFG Priority Program 1648 'Software for Exascale Computing', grant GO 1758/2-2


SPPEXA

# Justification of soft fault detection mechanism

- FAS correction problem

$$\mathbf{A}_{k-1}\tilde{v}_{k-1} = \mathbf{R}_{k-1}^{k} r_k + \mathbf{A}_{k-1}\tilde{v}_{k-1}^{(0)}$$

- Using linearity of the operator and $c_{k-1} = \tilde{v}_{k-1} - \tilde{v}_{k-1}^{(0)}$ yields

$$\|c_{k-1}\| \leq \|(\mathbf{A}_{k-1})^{-1}\| \|\mathbf{R}_{k-1}^{k}\| \|r_k\|$$

- $r_k = b_k - \mathbf{A}_k u_k^{(\nu)}$ on fine grid converges monotonously to zero if $\mathbf{A}$ is s.p.d.
- On coarser grid levels $b_k = \mathbf{R}_k^{k+1}(b_{k+1} - \mathbf{A}_{k+1}u_{k+1}^{(\nu)}) + \mathbf{A}_k \mathbf{I}_k^{k+1} u_{k+1}^{(\nu)}$ gives

$$\begin{aligned} r_k &= \mathbf{R}_k^{k+1}(b_{k+1} - \mathbf{A}_{k+1}u_{k+1}^{(\nu)}) + \mathbf{A}_k \mathbf{I}_k^{k+1} u_{k+1}^{(\nu)} - \mathbf{A}_k u_k^{(\nu)} \\ &= \mathbf{R}_k^{k+1}(b_{k+1} - \mathbf{A}_{k+1}u_{k+1}^{(\nu)}) + \mathbf{A}_k (u_k^{(0)} - u_k^{(\nu)}) \\ \Rightarrow \qquad \|r_k\| &\leq \|\mathbf{R}_k^{k+1}\| \|b_{k+1} - \mathbf{A}_{k+1}u_{k+1}^{(\nu)}\| + \|\mathbf{A}_k\| \, \|u_k^{(0)} - u_k^{(\nu)}\| \end{aligned}$$

- Operators are bounded, multigrid converges, smoothing property holds: $\|r_k\| \to 0$ and by this $\|c_k\| \to 0$