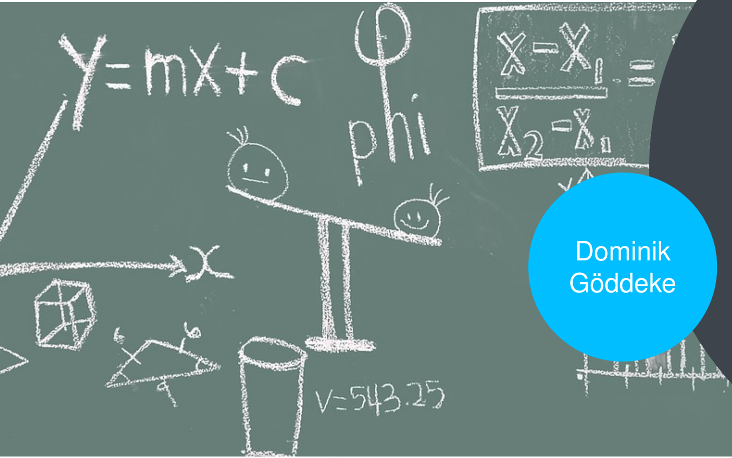
University of Stuttgart
Germany




# Enabling PDE software frameworks for exascale

PPAM 2017

September 2017

Dominik Göddeke

ians

SimTech
Cluster of Excellence

# Overview


Tagcloud of DFG call SPP-EXA 2012/2015, my weighting (generated using www.wordle.net)

# Hardware (r)evolution

## Parallelism, specialisation and heterogeneity

- Visible on all architectural levels now already
  - Fine-grained: SSE/AVX, GPU/MIC ‘threads’
  - Medium: GPUs, MICs, MC-CPUs, NUMA within nodes
  - Coarse: MPI between heterogeneous nodes
- Memory wall ever increasing limiter, despite NVM etc.
- Power is the root cause


## Consequences

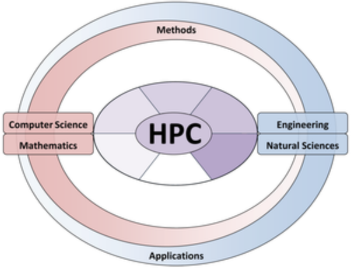
- Existing codes no longer run faster automagically
- Coordinated efforts needed to prepare simulation software for future systems
- In this talk: examples from Numerics and CSE for PDEs

Tombstone image shamelessly stolen from David Keyes

# SPPEXA overview

## DFG Priority Programme 1648: Software for Exascale Computing

- Largest (only?) HPC software initiative in Germany
- Stategically initiated, emphasis on both code development and methodological innovations
- Two funding periods, 2013–2015 and 2016–2018
- Overall budget 20+ M EUR
- Includes dedicated training at MSc/PhD level, dedicated software engineering practices, and 'built-in' collaborations between funded projects



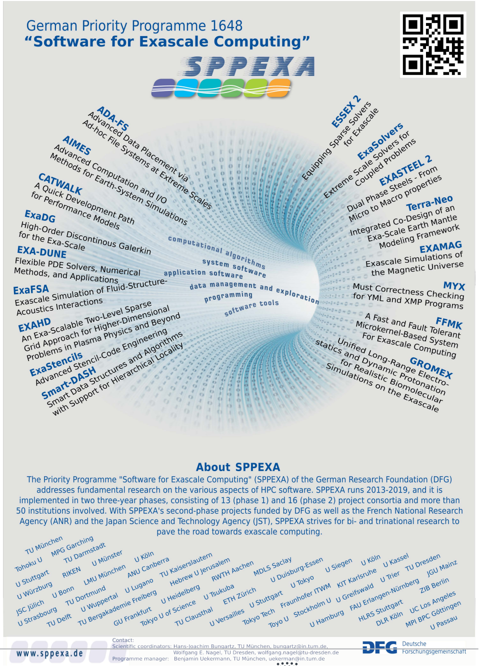
# SPPEXA overview

## Six research directions

- Computational algorithms
- System software
- Application software
- Data management and exploration
- Programming
- Software tools

## 13 projects in phase 1, 16 in phase 2

- Each addresses at least three areas,
- 4–7 co-PIs each at at least two sites
- Strong bi- and trilateral collaborations



# The EXA-DUNE project

P. Bastian, C. Engwer, D. Göddeke, O. Iliev,
O. Ippisch, M. Ohlberger, S. Turek

Universität Heidelberg — Zukunft. Seit 1386.
technische universität dortmund
Westfälische Wilhelms-Universität Münster
Fraunhofer ITWM
Universität Stuttgart
TU Clausthal — Clausthal University of Technology


**http://dune-project.org/**

# The EXA-DUNE project

## Starting point # 1: DUNE

- Berlin, Freiburg, Heidelberg, Münster, . . . , 100+ man-years
- Open-source flexible software framework / DSL, MPI-only
- Dimension-independent, different mesh types and FEs, hierarchical local refinement, separation mesh/linear algebra
- C++11, code generation, static polymorphism
- Main focus on flexibility and scalability through well-defined interfaces
- Applications: Navier-Stokes, Euler, Maxwell, elasticity, . . .

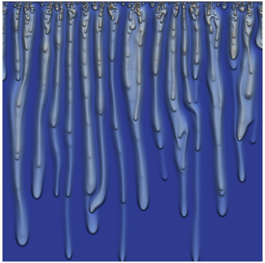
## Starting point # 2: FEA(S)T

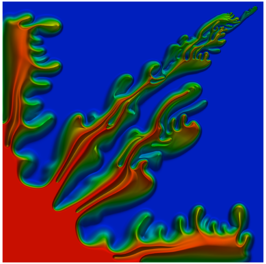
- Dortmund, Stuttgart
- Hardware-oriented numerics, e.g. mixed precision, locality
- Accelerators and multicores, node-level heterogeneity, MPI+X, . . .

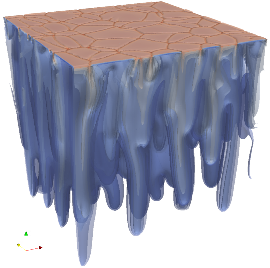
# The EXA-DUNE project

Project goal: Develop an open-source reusable and scalable software framework for the efficient numerical solution of PDEs

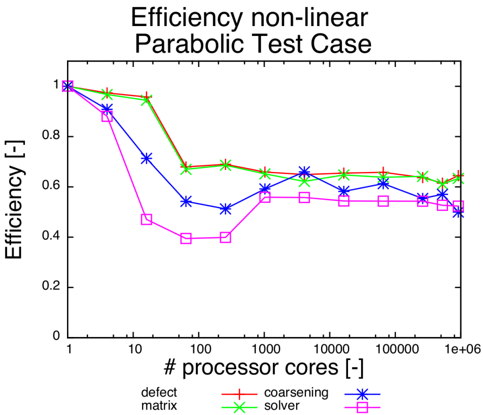
- Maintain flexibility, user-friendlyness and maintainability
- Improve performance and scalability under the hood
- By novel implementational and numerical techniques
- Two-phase porous media apps: solute transport, $CO_2$ sequestration, ...

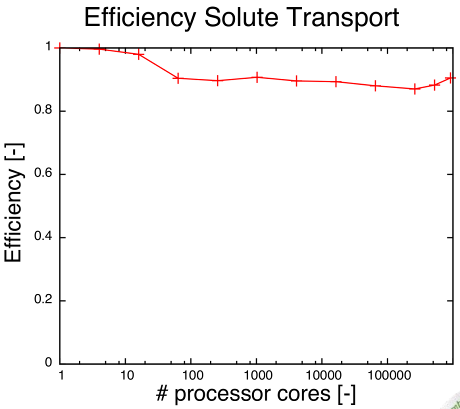






# Nice starting point




- JUQUEEN (Jülich), 917 504 hyperthreads, MPI-only

# Recent progress of the project

## Enhanced node-level performance

- Integration of GPUs and Phi behind a suitable abstraction layer
- Vectorisation of matrix assemby for unstructured-grid low-order schemes
- Application of sum-factorisation techniques for matrix-free high-order schemes (discontinous Galerkin)
- Combination of matrix-free and matrix-based solvers for flow and transport
- Hardware-aware preconditioning

## Resilience and asynchrony

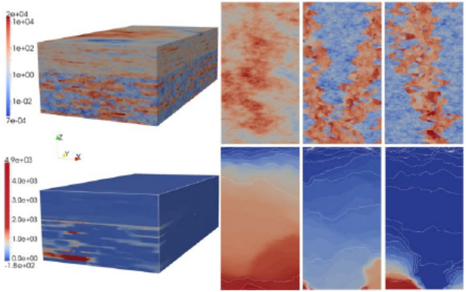
- Fault-tolerant multigrid
- ULFM-based protocol for local-failure-local-recovery scenarios
- Asynchronous abstraction layer based on C++ futures

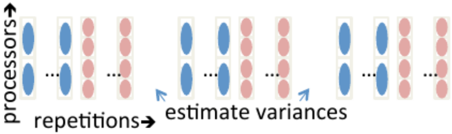
P. Bastian et al., Software for Exascale Computing – SPPEXA 2013–2015, Springer

# Recent progress of the project

## Additional levels of parallelism

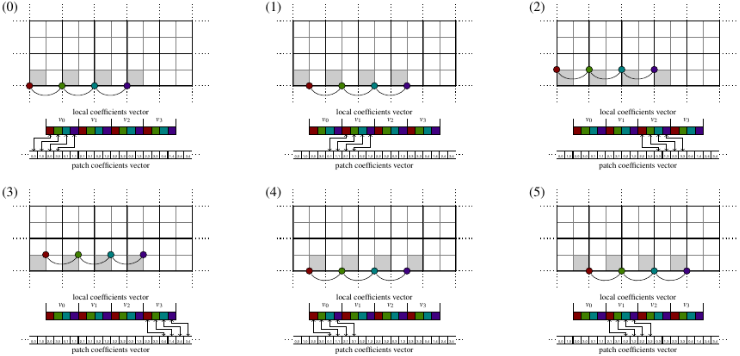
- Multiscale finite element methods
- Multilevel Monte Carlo and uncertainty quantification






P. Bastian et al., Software for Exascale Computing – SPPEXA 2013–2015, Springer

# Example: SIMD over multiple elements, low-order




| SIMD | lanes | thread | runtime | GFLOP/sec | % avail | % peak |
|---|---|---|---|---|---|---|
| none | 1 | 1 | 38.6 s | 3.0 | 19.2 | 0.6 |
| none | 1 | 16 | 2.5 s | 47.3 | 19.4 | 9.7 |
| AVX | 4 | 1 | 16.6 s | 5.0 | 32.0 | 1.0 |
| AVX | 4 | 16 | 1.1 s | 72.9 | 30.0 | 15.0 |

# Example: sum factorisation for high-order DG

- Exploit tensor product structure of base functions (and quadrature rules) to cleverly reorder operations
- Matrix free schemes can be implemented in $O(n^{d+1})$ instead of $O(n^{2d})$ operations per cell (polynomial order q and n=q+1)
- Algorithm by *Buis, Dyksen (1996)*
- Finite element application by e.g. *Melenk, Gerdes, Schwab (2001), Kronbichler, Korman (2012)*
- Memory demand per element is reduced since only 1D base functions need to be evaluated

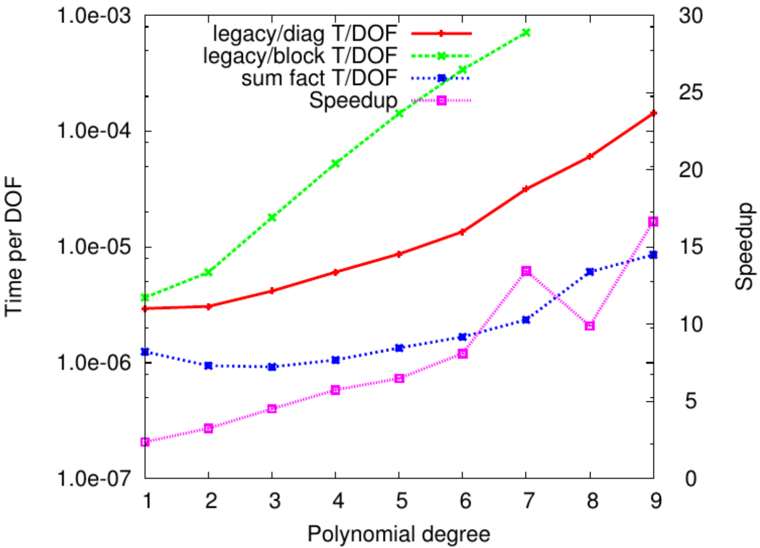
# Example: combined matrix matrix-free scheme

$$\nabla \cdot v = 0, \qquad v = -\frac{K}{\mu(c)}(\nabla p - \rho(c) g)$$

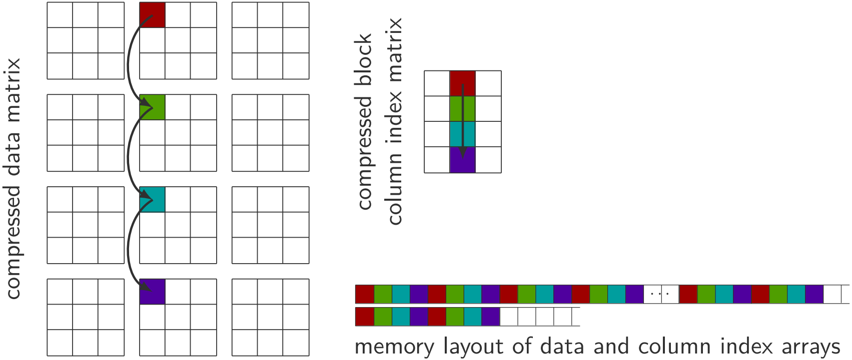
$$\partial_t(\Phi c) + \nabla \cdot (cv - D(v)\nabla c) = 0$$

- Operator splitting approach
- Flow: CCFV, two-point flux, AMG solver, $RT_0$ interpolation
- Transport:
  - Space: Weighted SIPG-DG (*Di Pietro, Ern, Guerm. 2008*)
  - Time: 2nd order explicit time stepping
  - Gauß-Lobatto tensor product basis
  - Under-integration of mass matrix (mass lumping)
    ➔ diagonal matrix
  - Sum Factorization during explicit time stepping for matrix free computation

# Example: sum factorisation for high-order DG

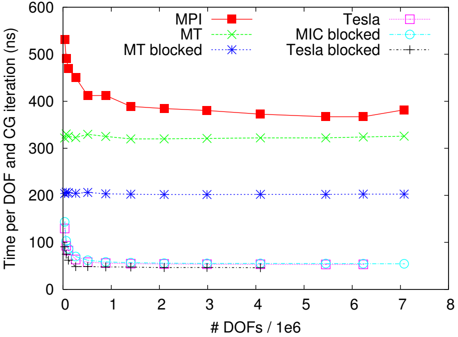


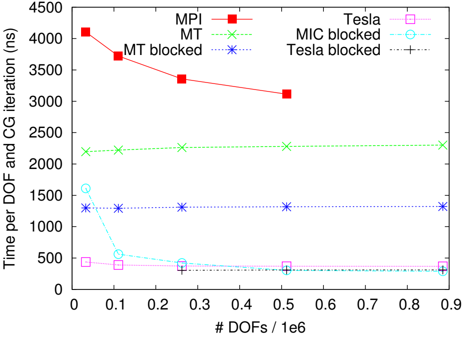
# Example: sparse linear algebra




- SELL-C-$\sigma$ format [Kreutzer et al., SISC, 2014]
- Extension to DG block sizes via horizontal vectorisation
- Extension to blocked matrices, saves bandwidth for index arrays

# Example: sparse linear algebra




- Stationary 3D diffusion, DG $p = 1$ and $p = 3$
- CG with point-Jacobi or block-Jacobi (precomputed inverses)
- Same format on all architectures

# Fault tolerance and resilience

# The resilience challenge

## Some important observations

- More components at exascale $\Rightarrow$ higher probability of failure
- Active debates to sacrifice reliability for energy efficiency
- Nightmare scenarios of MTBF $< 1\,\mathrm{h}$
- More importantly: very interesting research question in applied math

## Classical techniques

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

# Not only our approach: ABFT

## General concept: algorithm-based fault tolerance

- Exploit algorithmic properties to detect and correct faults
- Can be more efficient than middleware

## In this talk: some ideas for multigrid

- Self-stabilisation properties
- Target scenarios: from bitflips to node loss
- As much ABFT as possible, as little CPR as necessary
- Asynchronous, exponentially reduced checkpointing
- Black-box smoother protection from bitflips

# Self-stabilisation

**Theorem (Self-stabilisation)**

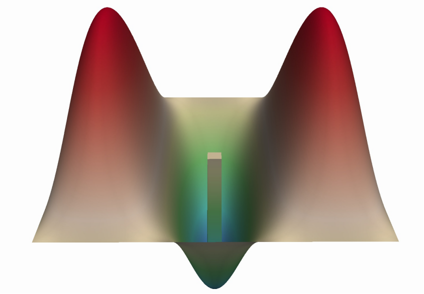
For single faults, multigrid is self-stabilising.

Qualitative proof:

- Realise that multigrid is a **linear fixed-point iteration**
- Assume no faults in data (matrix, discrete transfer operators)
- Consequence: contraction property of iteration operator holds
- Apply Banach's Fixed Point Theorem: convergence for any initial guess
- Realise that fault is just restart with new initial guess □

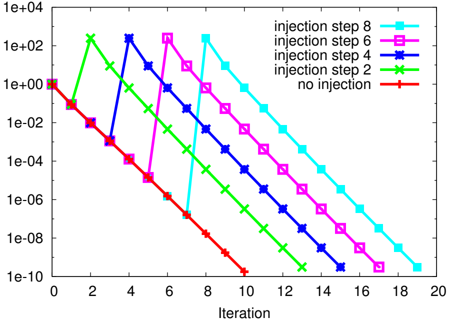
Problem solved. What happens quantitatively, i.e., not covered by numerics textbooks?

# Self-stabilisation



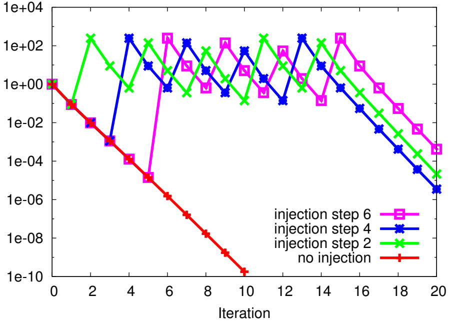
- Poisson problem $-\Delta u = f$ on $\Omega = [0, 1]^2$, Dirichlet BCs, Q2 FE, 1 M DOF
- V cycle geometric multigrid, residual-based convergence control
- Smooth problem: $f = -\Delta(\sin(\pi x)\sin(3\pi y))$
- Fault injection into some patch of fine grid iterate to emulate node loss

# Single fault injection at different iterations




- Convergence (residuals)
- Always convergence as proven, at most 2x iterations
- Large jump: fault injection implies a weak singularity at ‘fault boundary’ (steep change of curvature)

# Repeated fault injection at different locations




- Fault injection at alternating locations after every third iteration
- Good: MG converges nonetheless; Bad: MG only converges after fault injection has ended
- Indeed: MG only self-stabilising for infrequent faults

# Resilient multigrid with minimised checkpointing

## Main idea: explicitly exploit existing multigrid hierarchy

- Checkpoint: store last iterate on a coarser scale
- Restart: prolongate backup solution to fine scale
- Exploit exponentially decreasing data volume:
  $2^d$-fold savings per refinement level for conforming FEM

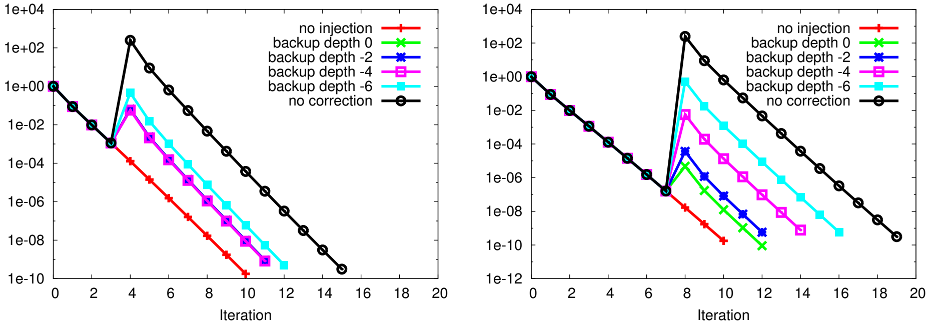
## Checkpoint-to-memory

- One extra down cycle (no smoothing), asynchronously
- Fault-free performance barely impacted

## Restart-from-memory

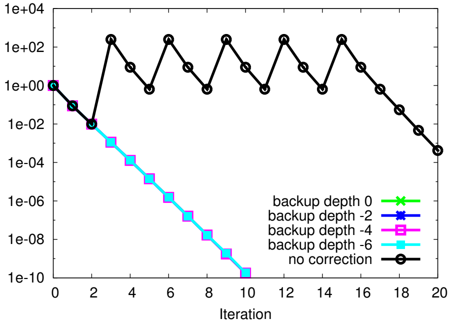
- Local prolongation on ‘backup rank’ or replacement node
- Implies P2P load imbalance instead of global sync as in CPR

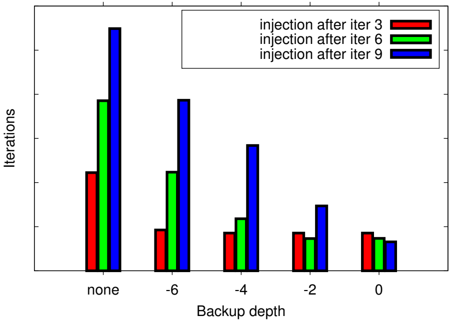
# Single fault injection and local repair




- Cyan plot corresponds to 4096x smaller checkpoint
- But no further restauration of convergence for faults in almost converged solution

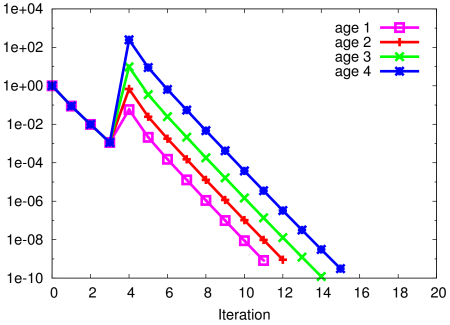
# Local auxiliary solve with checkpointed initial guess




- Solve aux. problem on lost patch using Dirichlet data from neighbours
- Convergence perfectly restored, but more expensive
- But: we can use the backup as an initial guess (!)
- 1.5x–4x less local iterations depending on backup depth

# Asynchronous checkpoints




- Backup depth 4 (256x), impact of checkpoint ‘age’
- Realistic delays: almost no impact
- Paves way for combination with next technique

# Silent data corruption

## Silent data corruption

- Soft transient faults $\Rightarrow$ wrong solutions, delayed convergence
- Sometimes noticeable a posteriori (divergence), mostly not
- Causes: radiation, smaller threshold voltage, silicon ageing, ...

## Core idea of our approach

- Use FAS (full approximation scheme) multigrid to increase robustness
- Based on nonlinear MG, so true approximation of the solution on each level and not just a correction
- Linear case: numerically equivalent, less than one fine SPMV overhead per cycle

# FAS multigrid

**FAS Multigrid prototype to solve $\mathbf{A}_h \mathbf{u}_h = \mathbf{f}_h$**

1. Smooth $\mathbf{u}_h$ on $\Omega_h$ with $\nu = 1, \ldots, 4$ Jacobi iterations
2. Compute $\mathbf{r}_h = \mathbf{f}_h - \mathbf{A}_h \mathbf{u}_h$
3. Restrict residuum and solution on $\Omega_{2h}$: $\hat{\mathbf{r}}_{2h} = \mathbf{R}_h^{2h} \mathbf{r}_h$, $\hat{\mathbf{u}}_{2h} = \mathbf{R}_h^{2h} \mathbf{u}_h$
   Update right hand side: $\hat{\mathbf{r}}_{2h} = \hat{\mathbf{r}}_{2h} + \mathbf{A}_{2h} \hat{\mathbf{u}}_{2h}$
4. Solve $\mathbf{A}_{2h} \mathbf{u}_{2h} = \hat{\mathbf{r}}_{2h}$
5. Correct solution on $\Omega_h$: $\mathbf{u}_h = \mathbf{u}_h + \mathbf{P}_{2h}^{h} (\mathbf{u}_{2h} - \hat{\mathbf{u}}_{2h})$
6. Smooth $\mathbf{u}_h$ on $\Omega_h$

# Black-box smoother protection

## Theoretical justification for the down-cycle

- Obvious: residual **r** converges to zero on finest grid
- Easy to prove: residual (monotonously) converges to zero on all grids

## Theoretical justification for the up-cycle

- Slightly nontrivial proof: correction vector **c** converges (monotonously) to zero on all grids

## Consequence: good fault indicators

- Both readily available without additional computation
- Applicable to both GMG and AMG
- In parallel: purely local, per-process indicators and thresholds

# Black-box smoother protection

## Practical realisation after smoothing on level $k$

- Compute index set $\mathcal{L}$ of possibly faulty components of $\mathbf{c}$ or $\mathbf{r}$ by comparing against level-specific threshold
- Extend by one (or few) layers of indices coupled by $\mathbf{A}$
- Replace faulty components by unsmoothed values (down-cycle), or by recomputed correction from (non-faulty) coarser correction, whichever is more recent
- Adaptively update threshold with data from *current cycle only*
- Two initialisations: first fine grid residuum and fault-free coarsest grid correction
- Scaled during level transfer with operator norm and/or tolerance factor
- Hierarchically coupled for F- and W-cycles

# Checksum protection for transfer stage

## Checksums

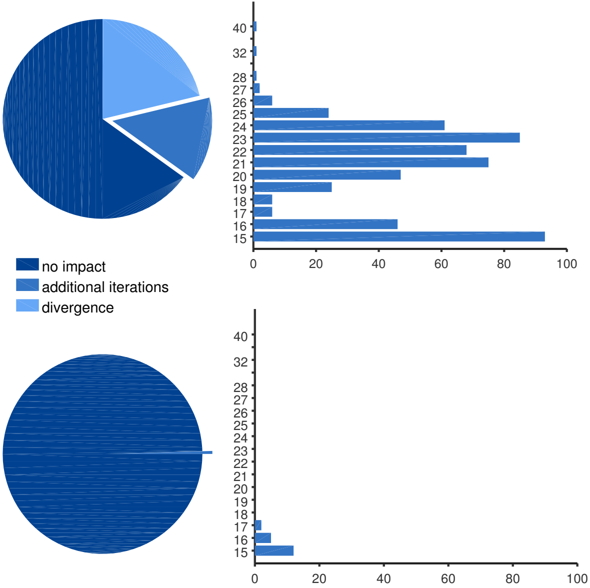
- Use identity $\mathbf{1}^\mathsf{T}(\mathbf{Ax} + \mathbf{y}) = (\mathbf{1}^\mathsf{T}\mathbf{A})\mathbf{x} + \mathbf{1}^\mathsf{T}\mathbf{y}$, precompute $\mathbf{1}^\mathsf{T}\mathbf{A}$ (column sums)
- Fault detection in $\mathbf{Ax} + \mathbf{y}$ by three dot products
- More elaborate schemes: detect and correct errors

## Combined approach

- Black-box smoother protection, checksums for the rest
- Walltime comparison, fault-free case, serial GMG
- FAS overhead 20 %, plus 10 % for FT

| | unprotected (MG) | unprotected (FAS) | transfer stage (checksums) | smoothing stage (new algorithm) | FTMG (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 |

# Numerical experiments



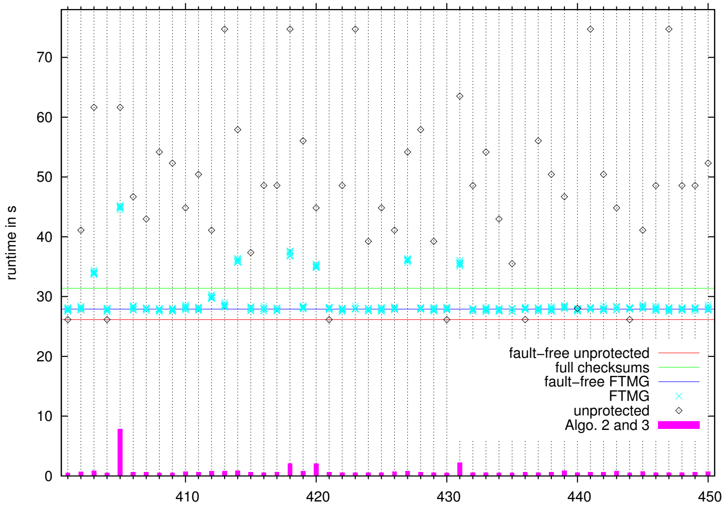

- Representative test problem:
  anisotropic diffusion
- Impact on classic (top) and fault-tolerant (bottom) multigrid algorithms
- For cases in which additional iterations are necessary the distribution of iteration numbers is shown on the right side
- In the fault-free scenario both algorithms need 14 iterations

# Numerical experiments

| V-cycle | **poisson** | **dico** | **andi** | **andicore** |
|---|---|---|---|---|
| **fault-free** #it | 4 | 6 | 14 | 7 |
| **classic** #it (div.) | 4.225 (272) | 6.268 (335) | 15.111 (850) | 7.466 (439) |
| **ftmg** #it | 4.038 | 6.007 | 14.007 | 7.017 |
| false-positives | 13 | 21 | 27 | 25 |
| worse | 15 | 1 | 0 | 1 |

- Statistics for V-cycle, 4000 different fault scenarios per test case
- Our approach always converges
- Very few false positives, which almost never lead to better iterations for the classic scheme

# Performance results



# Summary

# Summary of this talk

# Acknowledgements


- EXA-DUNE colleagues
- PhD students: Mirco Altenbernd, Malte Schirwon, Dirk Ribbrock
- EXC SimTech and SPPEXA for funding