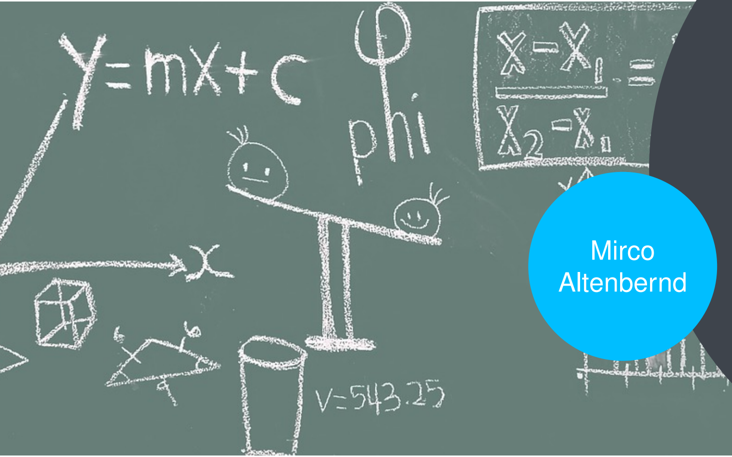
University of Stuttgart
Department of Mathematics



Multigrid and fault-tolerance

8th JLESC Workshop

April 17, 2018

Mirco Altenbernd

ians

SimTech

Motivation - Fault-tolerance

- More components at exascale $\Rightarrow$ higher probability of failure
- Active debates to sacrifice reliability for energy efficiency
- Nightmare scenarios of MTBF < 1 h

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

- 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 faster for large fault rates

Possible solution:
Exploit algorithmic properties to detect and correct faults on-the-fly (ABFT)

What we did

Compressed checkpointing for Multigrid

- Using inherent compression from multigrid to decrease checkpoint size
- Enables repair in node-loss scenario with good initial guess

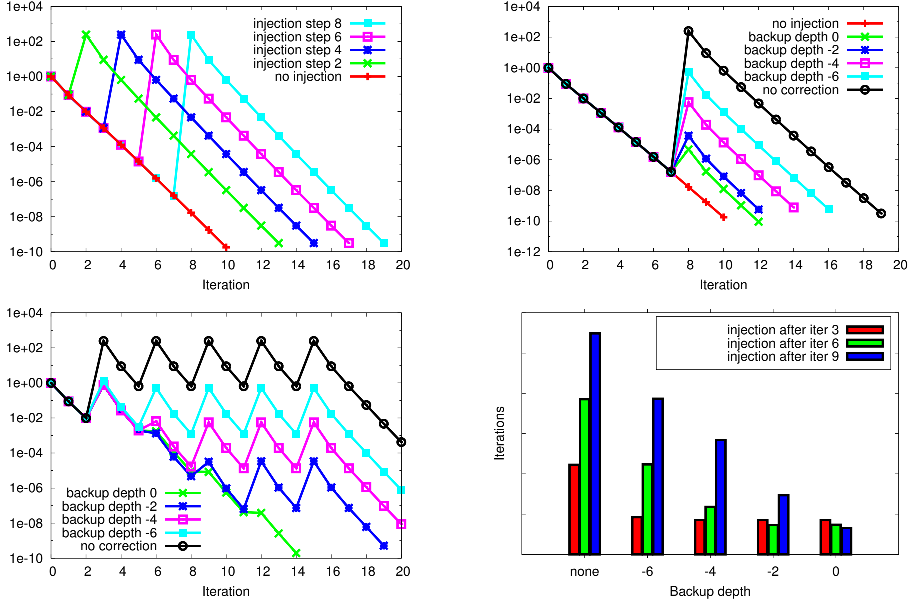
Fault-tolerant Multigrid

- Further increase multigrid's robustness with respect to bit-flips by using full approximation scheme
- Apply a local smoother protection to detect and repair soft faults

User level exception handling

- User-friendly `C++` MPI interface for parallel exception handling
- Propagate exceptions with MPI to always ensure same state on all ranks
- Ready for the *User level failure mitigation* proposal (ULFM)

Compressed checkpointing



Fault-tolerant Multigrid

- Switching from MG to FAS-MG allows additional SDC protection (FTMG)
- Numerical overhead of around 20%
- Protecting smoothing stage (> 80% of numerical operations)
- Repair faults with available resources from other levels

	poisson	**dico**	**andi**	**andicore**
fault-free	4	6	14	7
MG (div.)	4.225 (272)	6.268 (335)	15.111 (850)	7.466 (439)
FTMG	4.038	6.007	14.007	7.017

- Also working in parallel and with algebraic multigrid (AMG)

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

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)
- Adaptable to MPI-4 with ULFM (User-level failure-mitigation)

Code Example

```
try{ // scope to be protected
  Guard guard(communicator);
  do_computation();
  do_communication();
}catch(...) {
  // handle thrown exceptions
}
```

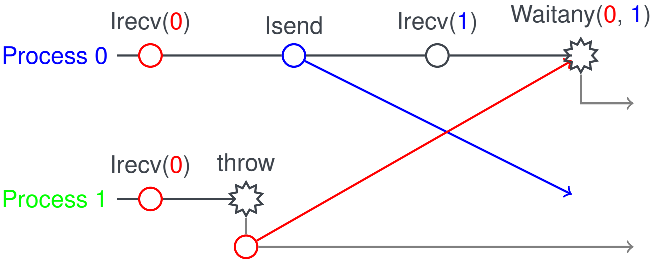
- Cheap guard object protects *try* block
- Is destructed during stack unwinding
- Propagate exception across communicator (uses `std::uncaught_exception`)

User level exception handling

MPI-3 variant

- Additional communication channel for exceptions
- Checked within each communication operation

⇒ Both processes are in the same state



MPI-4 variant

- Interface is adaptable to ULFM (proposed for MPI-4 standard)
- Provides functionality for
  - Hard fault *detection*
  - Communicator *revocation*
  - *Shrinking* of faulty communicator (i.e. excluding faulty processes)

⇒ Additional channel (`Irecv(0)`) is not needed anymore

What we want to do

- Integrating the new MPI interface into DUNE[1]
- Improving features/functionality of the interface for wider applicability
- Evaluating and combining developed concepts
  - Asynchronous checkpointing for compressed checkpoints
  - Asynchrony in multigrid:
    Local smoothing while restoring lost processors?
  - Multigrid as preconditioner:
    Compressed checkpointing for outer solver with MG hierarchy?
  - ...

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

Ideas for concrete cooperation

Fault-tolerance

- How to protect the assembly procedure?
- Other options to secure matrix-vector multiplication than checksums?
- How to ensure correctness of matrix-free operators?
- …

Asynchrony

- Asynchrony in multigrid methods?
- Concepts for asynchronous checkpointing?
- …

Jointly apply our techniques to your linear solvers?

Further questions

- Do you anticipate/have you seen reasons for FT?
- What types/frequencies of failures/faults are you expecting in future exascale systems?
- How to evaluate/simulate fault-tolerant methods in a serious way?
- How would your schemes break if you can no longer assume receiving correct results?
- What functions do you expect from a fault-tolerant C++ MPI interface with exception handling?