Fault-tolerant parallel multigrid
ECCM-ECFD 2018 - MS116A
June 14, 2018

Mirco Altenbernd
& Dominik Gödeke
Motivation - Fault-tolerance

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

<table>
<thead>
<tr>
<th>#cores</th>
<th>1</th>
<th>100</th>
<th>10,000</th>
<th>1,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTBF</td>
<td>5 years</td>
<td>18 days</td>
<td>4 hours</td>
<td>3 mins</td>
</tr>
</tbody>
</table>

- 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

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

1. **Compressed checkpointing for Multigrid**
   - Using inherent compression from multigrid to decrease checkpoint size
   - Enables repair in node-loss scenario with good initial guess

2. **Fault-tolerant Multigrid**
   - Further increase multigrid’s inherent robustness with respect to bit-flips by using full approximation scheme multigrid
   - Apply a local smoothing stage protection to detect and repair soft faults

3. **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

Evaluation
- 2D poisson problem
- V-cycle multigrid, jacobi smoother
- Node-loss simulation:
  Set values in a small area to zero

Repair
- Restore lost data with compressed checkpoint
- Compression via MG transfer operator
- Data reduction in $d$ dimensions:
  $d^n$ per level (backup depth)
Compressed checkpointing

**Problem**
- Recurrent and late node-losses need less compressed checkpoints
- At the end no compression is possible

**Solution**
- Solve an auxiliary problem with dirichlet boundary to improve restoration
- Use compressed data as initial guess ⇒ Reduces iteration count significantly
- Alternative: Other compression techniques like SZ

---

1 D. Tao, S. Di, Z. Chen and F. Cappello, [...] Lossy Compression for Scientific Data Sets [...] Computing Research Repository, 2017
Compressed checkpointing

Overview
- MG compressed checkpoints can be used to recover from node-losses
- Early on high compressed data is sufficient
- Later compression rate has to be decreased
- Eventually an auxiliary problem has to be solved or another compression technique has to be used
- The compressed data is a good initial guess for the auxiliary problem

Project was finished but we have got new ideas:
- Using MG transfer operators to compress data of an outer solver
- Compare compression quality/runtime to different lossy-compression techniques like SZ compression

Fault-tolerant Multigrid

Aim
Fault-tolerant with respect to silent data corruption

Observation
Most time is spent within the smoothing stage

Idea
- Use invariants of *Full Approximation Scheme* MG (FASMG) to test output of smoothing stage
- Don’t ensure correctness value by value
- Only verify if output is ‘good enough’
- Protect remaining part with checksums
Fault-tolerant Multigrid

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

⇒ Overall overhead of ~ 30% compared to classical MG

<table>
<thead>
<tr>
<th></th>
<th>unprotected (MG)</th>
<th>unprotected (FASMG)</th>
<th>transfer stage (checksums)</th>
<th>smoothing stage (new algorithm)</th>
<th>FTMG (both)</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>35.49</td>
<td>43.02</td>
<td>45.23</td>
<td>44.76</td>
<td>46.18</td>
</tr>
<tr>
<td>factor</td>
<td>0.825</td>
<td>1</td>
<td>1.051</td>
<td>1.040</td>
<td>1.073</td>
</tr>
<tr>
<td>factor</td>
<td>1</td>
<td>1.212</td>
<td>1.274</td>
<td>1.261</td>
<td>1.301</td>
</tr>
</tbody>
</table>
Fault-tolerant Multigrid

Protection and Repair mechanism

- Calculate thresholds based on output of smoothing stage
- Transfer them to next grid level
- Check values of next smoothing stage against them
- If not sufficient replace values by values from previous level

<table>
<thead>
<tr>
<th></th>
<th>poisson</th>
<th>dico</th>
<th>andi</th>
<th>andicore</th>
</tr>
</thead>
<tbody>
<tr>
<td>fault-free</td>
<td>4</td>
<td>6</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>MG (div.)</td>
<td>4.225 (272)</td>
<td>6.268 (335)</td>
<td>15.111 (850)</td>
<td>7.466 (439)</td>
</tr>
<tr>
<td>FTMG</td>
<td>4.038</td>
<td>6.007</td>
<td>14.007</td>
<td>7.017</td>
</tr>
<tr>
<td>false-positives</td>
<td>13</td>
<td>21</td>
<td>27</td>
<td>25</td>
</tr>
</tbody>
</table>

Iteration count with approximately two SDC every iteration.

Fault-tolerant Multigrid

Applicability

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

<table>
<thead>
<tr>
<th>#it</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>25</th>
<th>34</th>
<th>41</th>
<th>div</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMG</td>
<td>97</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td>87</td>
<td>17.72</td>
</tr>
<tr>
<td>FTAMG</td>
<td>179</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>17.12</td>
</tr>
</tbody>
</table>

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

- Different cycle types:

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

```cpp
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
Recap

- We developed three ‘orthogonal’ approaches to increase fault-tolerance, especially for multigrid algorithms:
  - Efficient SDC protection with built-in properties
  - Partial restoration with compressed checkpoints for node-losses
  - Exception-propagation to ensure the same state in MPI programs
- Combination is still in progress but first tests seem promising
- ’User level exception handling’ can be used for many algorithms to develop strategies for fault-tolerance in MPI-3 and is adaptable to MPI-ULFM
- Currently evaluating the advantages of MG compression and SZ compression\(^2\)

\(^2\) Initiated cooperation with Jon Calhoun (Clemson University, South Carolina, USA)
What’s next?

- Integrating the new MPI interface into DUNE\(^3\)
- 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
    - Adaptive combination with SZ compression
  - …

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

\(^3\)funded by DFG: German Priority Programme 1648, SPPEXA, EXADUNE
Thank you for your attention!

- Dominik Göddeke (University of Stuttgart)
- Nils-Arne Dreier and Christian Engwer (University of Münster)

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