Multigrid and fault-tolerance
8th JLESC Workshop
April 17, 2018

Mirco Altenbernd
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

<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 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)
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

<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>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>#it</th>
<th>div</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMG</td>
<td>17</td>
<td>87</td>
<td>17.72</td>
</tr>
<tr>
<td>FTAMG</td>
<td>179</td>
<td>0</td>
<td>17.12</td>
</tr>
</tbody>
</table>
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 $(\text{Irecv}(0))$ 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?