ULFM-MPI Implementation of a Resilient Task-Based Partial Differential Equations Preconditioner

F. Rizzi†, K. Morris†, K. Sargsyan†, P. Mycek‡, C. Safta†, O. LeMaitre‡, O. Knio‡, B. Debusschere†

† Sandia National Laboratories, Livermore, CA
‡ Duke University, Durham, NC

FTXS16 Workshop - HPDC16
– May 2016 –

Supported by the US Department of Energy (DOE)
Advanced Scientific Computing Research (ASCR)

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.
Resilence

Collection of techniques to keep applications running to a correct solution in a timely and efficient manner *despite* underlying system faults.


Do We Need Resilience?

Resilience

Collection of techniques to keep applications running to a correct solution in a timely and efficient manner *despite* underlying system faults.


IEEE, Feb.2016, Al Geist (ORNL)

“As a child, were you ever afraid that a monster lurking in your bedroom would leap out of the dark and get you?”

“Lack of resilience is a similar monster, hiding in the steel cabinets of the supercomputers and threatening to crash the largest computing machines.”
**Terminology**

- **Fault**: the cause of an error
  - active/inactive fault cause/not cause errors.
  - generally local to a single component.
  - e.g. a cracked wire inside a cable.

- **Error**: the part of the state that may lead to a failure
  - may propagate from component to component.
  - e.g. incorrect bit flip during transmission caused by wire.

- **Failure**: transition to incorrect service
  - transition from correct service to incorrect service.
  - e.g. incorrect bit may lead to wrong result.

Broadly Speaking

- **Hard**: Activation is systematically reproducible.
- **Soft**: Activation is not systematically reproducible.
- **Active**: Fault causes an error.
- **Dormant**: Fault does not cause an error. The dormant fault is activated when it causes an error.
- **Permanent**: Presence is continuous in time.
- **Transient**: Presence is temporary.
- **Intermittent**: Fault is transient and reappears.

Addressing Failures in Exascale Computing, Snir et al., 2012.
How Many Bit Flips At Petascale?

- Jaguar: Cray XT5 system at Oak Ridge (until 2012).
- Jaguar had 360 terabytes of main memory, all protected by ECC.
- Guess how often Jaguar had a bit spontaneously change state?
How Many Bit Flips At Petascale?

- Jaguar: Cray XT5 system at Oak Ridge (until 2012).
- Jaguar had 360 terabytes of main memory, all protected by ECC.
- Guess how often Jaguar had a bit spontaneously change state?
  \[ \sim 300 \text{ per minute!} \]
- At exascale error rates will become larger.
  - Hardware failures are expected to be more frequent.
  - More complex hardware \( \Rightarrow \) more complex, error-prone software.
  - Application codes are becoming more complex.

---

\[ ^a \text{Al Geist, IEEE Spectrum, 2016} \]

\[ ^b \text{A Field Study of DRAM Errors, V.Sridharan, D.Liberty, 2012} \]
Challenges & Needs

- Hypothetically hardware can take care of most of it...
- ...at the expense of energy consumption, money, and asynchrony.
- Power is a big obstacle towards exascale.
- High tradeoffs between power, resiliency and performance.
  For instance: if an application can tolerate memory bit flips for certain parts of its memory, it can ask the OS to turn off ECC checks for those memory regions and potentially improve power and performance.
- Cross-cutting research needed to explore these areas.
<table>
<thead>
<tr>
<th>Motivation</th>
<th>Algorithm</th>
<th>Implementation</th>
<th>Results</th>
<th>Conclusions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Algorithm</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*F. Rizzi*  
Resilient Approach for PDE  
Min:
## Resilient Extreme Scale Scientific Simulations (REXSSS)

### What?

Domain-decomposition-based preconditioner for PDEs.
- Currently for elliptic equations (1D, 2D).
- Other PDEs (in progress).

### How?

Recasting the original PDE problem as a sampling problem, followed by a resilient data manipulation to achieve the final solution update.

### Why?

- We do not characterize all types of system faults that can occur, but focus solely on the information that a simulation provides.
- Target: silent data corruptions (SDCs) and hard faults (nodes/cores failing).
1D problem:

\[
\begin{align*}
\mathcal{L} y(x) &= g(x), \quad \text{in } \Omega = (x_-, x_+) \\
y(x_-) &= y^-,
y(x_+) &= y^+,
\end{align*}
\]

\(\mathcal{L}\) is a **linear**, elliptic operator.

The solution at point \(x_0\) **linearly depends** on the boundary conditions:

\[y(x_0) = a + by^- + cy^+\]
Algorithm Overview: Domain Decomposition & Boundary Maps

- Grid with current state.
- Partition space with overlapping subdomains.
- Treat subdomains independently.
- Define sampling range for each boundary. Sample and solve PDE locally.
- From samples build maps:
  \[ y_1 = f(y_2, y_L) = a + by_2 + cy_L \]
  \[ y_2 = g(y_1, y_R) = d + ey_1 + fy_R \]
- Maps link the subdomains.
- \( y_L, y_R \) are known BC.
- Solve, new state: \( (y_1^*, y_2^*) \).
- If needed: update range, repeat loop.
Algorithm Overview: Domain Decomposition & Boundary Maps

- Grid with current state.
- Partition space with overlapping subdomains.
- Treat subdomains independently.
- Define sampling range for each boundary. Sample and solve PDE locally.
- From samples build maps:
  \[ y_1 = f(y_2, y_L) = a + by_2 + cy_L \]
  \[ y_2 = g(y_1, y_R) = d + ey_1 + fy_R \]
  Maps link the subdomains.
- \( y_L, y_R \) are known BC.
- Solve, new state: \((y_1^*, y_2^*)\).
- If needed: update range, repeat loop.
Algorithm Overview: Domain Decomposition & Boundary Maps

- Grid with current state.
- Partition space with overlapping subdomains.
- Treat subdomains independently.
- Define sampling range for each boundary. Sample and solve PDE locally.
- From samples build maps: $y_1 = f(y_2, y_L) = a + by_2 + cy_L$ $y_2 = g(y_1, y_R) = d + ey_1 + fy_R$
- Maps link the subdomains.
- $y_L, y_R$ are known BC.
- Solve, new state: $(y_1^*, y_2^*)$.
- If needed: update range, repeat loop.
Algorithm Overview: Domain Decomposition & Boundary Maps

- Grid with current state.
- Partition space with overlapping subdomains.
- Treat subdomains independently.
- Define sampling range for each boundary. Sample and solve PDE locally.
- From samples build maps: 
  \[ y_1 = f(y_2, y_L) = a + by_2 + cy_L \]
  \[ y_2 = g(y_1, y_R) = d + ey_1 + fy_R \]
- Maps link the subdomains.
- \( y_L, y_R \) are known BC.
- Solve, new state: \( (y_1^*, y_2^*) \).
- If needed: update range, repeat loop.
Algorithm Overview: Domain Decomposition & Boundary Maps

- Grid with current state.
- Partition space with overlapping subdomains.
- Treat subdomains independently.
- Define sampling range for each boundary. Sample and solve PDE locally.
- From samples build maps:
  \[ y_1 = f(y_2, y_L) = a + by_2 + cy_L \]
  \[ y_2 = g(y_1, y_R) = d + ey_1 + fy_R \]
  Maps link the subdomains.
- \( y_L, y_R \) are known BC.
- Solve, new state: \((y_1^*, y_2^*)\).
- If needed: update range, repeat loop.
Algorithm Overview: Domain Decomposition & Boundary Maps

- Grid with current state.
- Partition space with overlapping subdomains.
- Treat subdomains independently.
- Define sampling range for each boundary. Sample and solve PDE locally.
- From samples build maps:
  \[ y_1 = f(y_2, y_L) = a + by_2 + cy_L \]
  \[ y_2 = g(y_1, y_R) = d + ey_1 + fy_R \]
  Maps link the subdomains.
- \( y_L, y_R \) are known BC.
- Solve, new state: \( (y_1^*, y_2^*) \).
- If needed: update range, repeat loop.
Algorithm Overview: Domain Decomposition & Boundary Maps

- Grid with current state.
- Partition space with overlapping subdomains.
- Treat subdomains independently.
- Define sampling range for each boundary. Sample and solve PDE locally.
- From samples build maps:
  \[ y_1 = f(y_2, y_L) = a + by_2 + cy_L \]
  \[ y_2 = g(y_1, y_R) = d + ey_1 + fy_R \]
- Maps link the subdomains.
- \( y_L, y_R \) are known BC.
- Solve, new state: \((y_1^*, y_2^*)\).
- If needed: update range, repeat loop.
Extension

\[ y_1 = f_1(y_L, y_2) \]
\[ y_2 = f_2(y_1, y_4) \]
\[ y_3 = f_3(y_1, y_4) \]
\[ y_4 = f_4(y_3, y_6) \]
\[ y_5 = f_5(y_3, y_6) \]
\[ y_6 = f_6(y_5, y_R) \]
Extension

Stage 1: discretization

Workflow
Extension

Stage 1: discretization

Stage 2: partitioning

Workflow
Extension

Stage 1: discretization

Stage 2: partitioning

Stage 3: state & range

Workflow
Extension

Stage 1: discretization
Stage 2: partitioning
Stage 3: state & range
Stage 4: sampling
Extension

Stage 1: discretization

Stage 2: partitioning

Stage 3: state & range

Stage 4: sampling

Stage 5: regression, boundary maps

Workflow

Sampling range

PDE samples

State
Extension

Stage 1: discretization

Stage 2: partitioning

Ax = b
Stage 6: solve boundary maps system

Stage 3: state & range

Stage 4: sampling

Stage 5: regression, boundary maps

Sampling range

PDE samples

Workflow
Extension

Stage 1: discretization

Stage 2: partitioning

Stage 3: state & range

Stage 4: sampling

Stage 5: regression, boundary maps

Stage 6: solve boundary maps system

Stage 7: update state & range

Ax=b

Sampling range

PDE samples

State
Robust Regression: Resilience to SDCs

- High-dim boundary maps:
  \[ u(x^*) = a + bu_1 + cu_2 + \ldots \]

- Generate samples, suppose some are corrupted, run regression.

- \( \ell_1 \) noise model is robust against presence of corrupted data.
Implementation
Server/Client-based Implementation

- Cluster: 1 server + n clients.
- Servers:
  - Communicate between each other.
  - Safe data/state storage (sandboxed).
- Clients:
  - Independent from one another.
  - Only serve as computing units.
- Separates state from computation: reduces the overall vulnerability.
- Fault-tolerance supported via ULFM-MPI: resilient to MPI ranks crashing.
- Resilient to clients crashing because even if tasks are lost, state is safe.
- It aligns with the vision of future exascale architectures involving heterogeneous and hierarchical hardware required to meet energy and cost constraints.
- C++ code, two external dependencies: Trilinos and Boost.
ULFM and SCM: Why is this a good combination?

- Hard faults modeled as clients crashing.
- Server simply continues the execution using only the clients that are alive.
  - √ Avoid ULFM collective procedures to rebuild the client/communicators.
- Servers probe the corresponding cluster communicator using \texttt{MPI\_ANY\_SOURCE} to assess whether a new message is arriving from one of the clients.
- Client crash:
  1. tell ULFM we are aware of the failure.
  2. Fix communicators, etc.
  3. continue normal execution.
• Edison (NERSC), Cray-MPICH.
• Elliptic PDE on unit square.
• Subdomain size: $180^2$.
• 64 clients per server, each client has 4 MPI ranks.
• Occupancy: 99% clients, 1% servers.
• Scaling up to 115 $k$.
  • Weak efficiency: $\sim 93\%$.
  • Strong efficiency: $\sim 90\%$. 
Resilience Results
Test Problem

- 2D linear elliptic equation.
- $201^2$ grid, 3x3 subdomains.
- Nominally: 3249 sampling and 2136 regression tasks.
- 1 server, 14 clients size 2.
- Faults affect clients only.
Injecting SDC and Hard Faults

Selective reliability

Inject/perturb applications at target points and evaluate how it behaves. Some parts of the algorithm are assumed to be handled in a more reliable manner than others. M.Hoemmen,M.Heroux,2012

Silent Data Corruptions (SDC)

- Selective reliability: only affect sampling stage.
- Injection at random; \( #\text{faults} = 0.25, 0.5, 1\% \) of tasks.
- Corrupt all boundary conditions data of a task.
- Bit-flip model: random bit-flip in binary representation.

Hard Faults

- Selective reliability: can affect sampling and regression.
- Injection at random; 2, 4, 6 clients crashing.
- Actually kill the processes associated with those ranks.
Injecting SDC and Hard Faults

Selective reliability

Inject/perturb applications at target points and evaluate how it behaves. Some parts of the algorithm are assumed to be handled in a more reliable manner than others. M.Hoemmen,M.Heroux,2012

Silent Data Corruptions (SDC)

- Selective reliability: only affect sampling stage.
- Injection at random; \( \# \text{faults} = 0.25, 0.5, 1\% \) of tasks.
- Corrupt all boundary conditions data of a task.
- Bit-flip model: random bit-flip in binary representation.

Hard Faults

- Selective reliability: can affect sampling and regression.
- Injection at random; 2, 4, 6 clients crashing.
- Actually kill the processes associated with those ranks.
Injecting SDC and Hard Faults

Selective reliability

Inject/perturb applications at target points and evaluate how it behaves. Some parts of the algorithm are assumed to be handled in a more reliable manner than others. M. Hoemmen, M. Heroux, 2012

Silent Data Corruptions (SDC)

- Selective reliability: only affect sampling stage.
- Injection at random; \#faults = 0.25, 0.5, 1% of tasks.
- Corrupt all boundary conditions data of a task.
- Bit-flip model: random bit-flip in binary representation.

Hard Faults

- Selective reliability: can affect sampling and regression.
- Injection at random; 2, 4, 6 clients crashing.
- Actually kill the processes associated with those ranks.
Silent Data Corruptions (SDC)
- Resilience condition: out of the samples used in the regression, the number of uncorrupted samples has to be greater than the minimum set needed to have a well-posed regression problem.

Hard Faults
- Server continues the execution using only the clients that are alive.
- No need for ULFM collectives to rebuild broken communicators.

Oversampling
- Oversampling: $\rho > 1$, such that $N = \rho N_{nom}^s$.
- $N_{nom}^s$: number of samples for the fault-free scenario.

- Analyze hard faults only.
- Hard and soft faults together.
• Angular direction = client name.
• Data = total number of tasks being handled during the simulation.
• No-fault case: workload is fairly uniform
• As expected, increasing the number of faults causes the clients that are alive to handle more and more tasks to compensate for those that are dead.
• the best case scenario is when all faults affect regression because full computational power is available for a longer part of the simulation
• HF for both: losing 14%, 28% and 42% of the clients yields, respectively, a total overhead of 8%, 19% and 30%.
• Consider 4 hard faults; *four-fold* increase in SDC from 9 to 33 causes the sampling overhead to only increase from 9% to about 15%.
• Regression overhead only increases from 30% to about 38%.
• This yields the total overhead to only increase from 21% to 28%.
• Application is resilient to:
  • Silent Data Corruptions during sampling.
  • Missing data due to communication issues or node failures.

• Sampling/decomposition approach provides concurrency/parallelism.

• Convergence is achieved in all cases.

• Scalability is excellent.

• Ongoing work/outlook:
  • Dimensionality reduction.
  • Extension to other types of PDE.
ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award Number 13-016717.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-AC04-94AL85000.

THANK YOU!
This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award Number 13-016717.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-AC04-94AL85000.

THANK YOU!