OAK RIDGE National Laboratory

Performance Efficient Multiresilience using Checkpoint Recovery in Iterative Algorithms

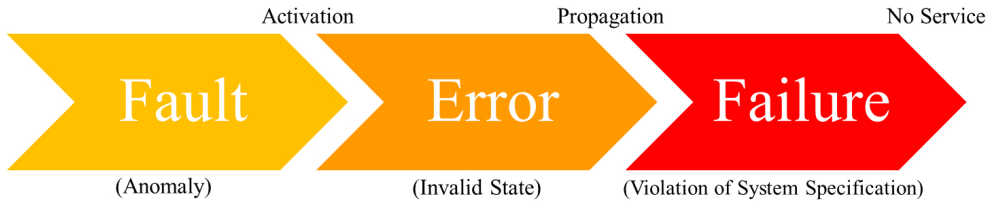
Rizwan A. Ashraf and Christian Engelmann,
Computer Science and Mathematics Div.,
Oak Ridge National Laboratory (ORNL), USA.

11th Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids,

Turin, Italy, 28th August, 2018.

Multiresilience

- ***Resilience*** in high performance computing (HPC) applications: *the ability of the applications to efficiently handle errors and recover from failures, while maintaining forward progress as desired by the user.*
- HPC Applications are affected by multiple types of errors which hinders with their ability to make forward progress and their correctness.
  - **Soft Errors:** transient in nature, e.g., silent data corruptions (SDC),
  - **Hard Errors:** temporarily or permanently unavailable resource, e.g., process failures.



Motivation & Scope

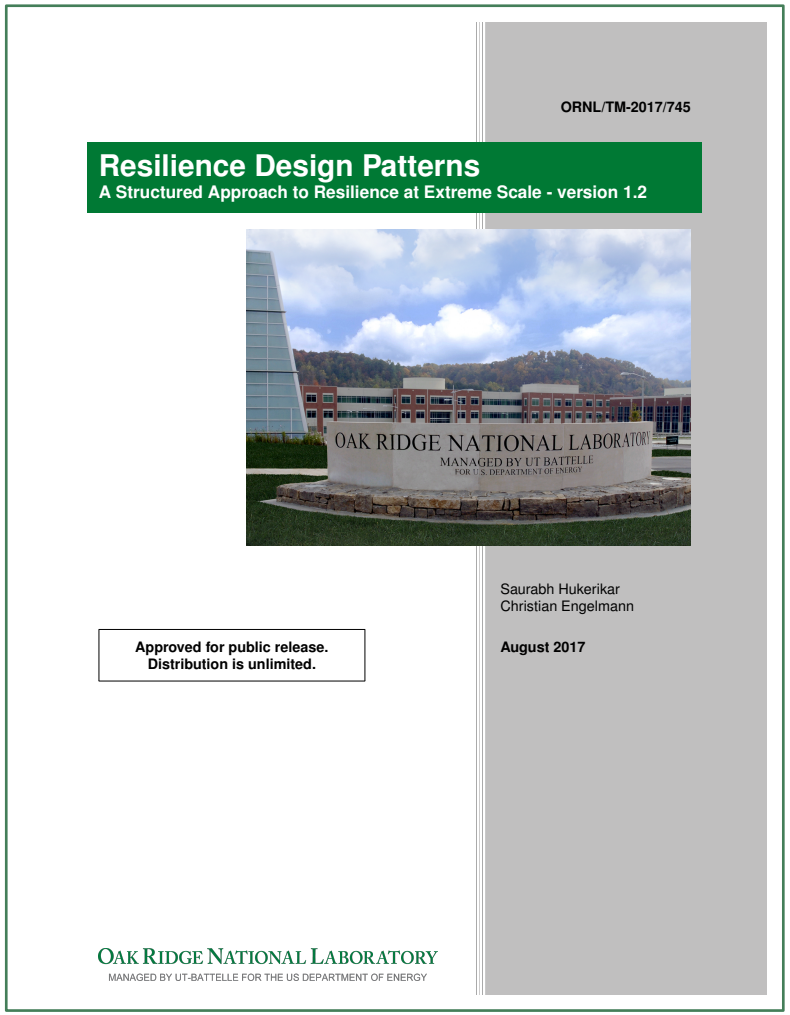
- **Scope:** iterative HPC applications which can tolerate soft errors by taking additional time to converge to a solution [1][2].
- Proposed ***performance models for resilience design patterns*** which serve as a guide for users to build optimal and efficient applications.
- Models consider the interaction b/w *non-ideal* soft error detection and checkpoint-based recovery.
- Use of models to compare the impact on time-to-solution b/w:
  - Inaccurate and low overhead soft-error detection —vs—
  - High Accuracy and high overhead soft-error detection.

Design Patterns for Resilience

- *Systematically* integrate multiple techniques to detect and handle multiple error and failure events, with minimal impact on performance.
- Patterns provide *generalizable solutions* to recurring problems.
- Patterns *do not* provide concrete solutions, instead focus on a reproducible strategy which may be used many times, implemented in different manners.
- Multiple patterns are instantiated across layers of the system stack, interlinked using a *building blocks approach*.

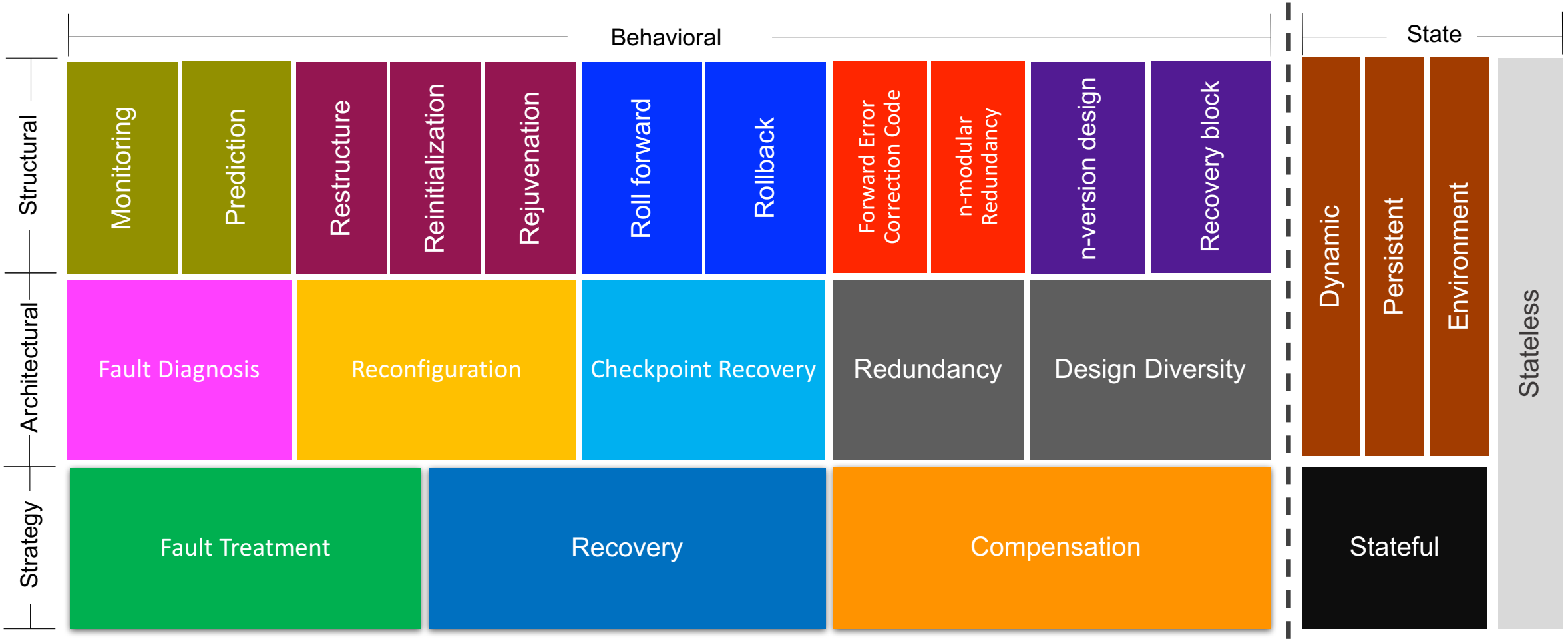
Resilience Design Patterns Specification v1.2

- Taxonomy of resilience terms and metrics
- Classification of resilience design patterns
- Catalog of resilience design patterns
  - Uses a pattern language to describe solutions
  - 3 strategy patterns, 5 architectural patterns, 11 structural patterns, and 5 state patterns
- Case studies using the design patterns
- A resilience design spaces framework



Saurabh Hukerikar and Christian Engelmann. **Resilience Design Patterns: A Structured Approach to Resilience at Extreme Scale (Version 1.2)**. Technical Report, ORNL/TM-2017/745, Oak Ridge National Laboratory, Oak Ridge, TN, USA, August, 2017. DOI: 10.2172/1436045

Resilience Design Patterns Classification

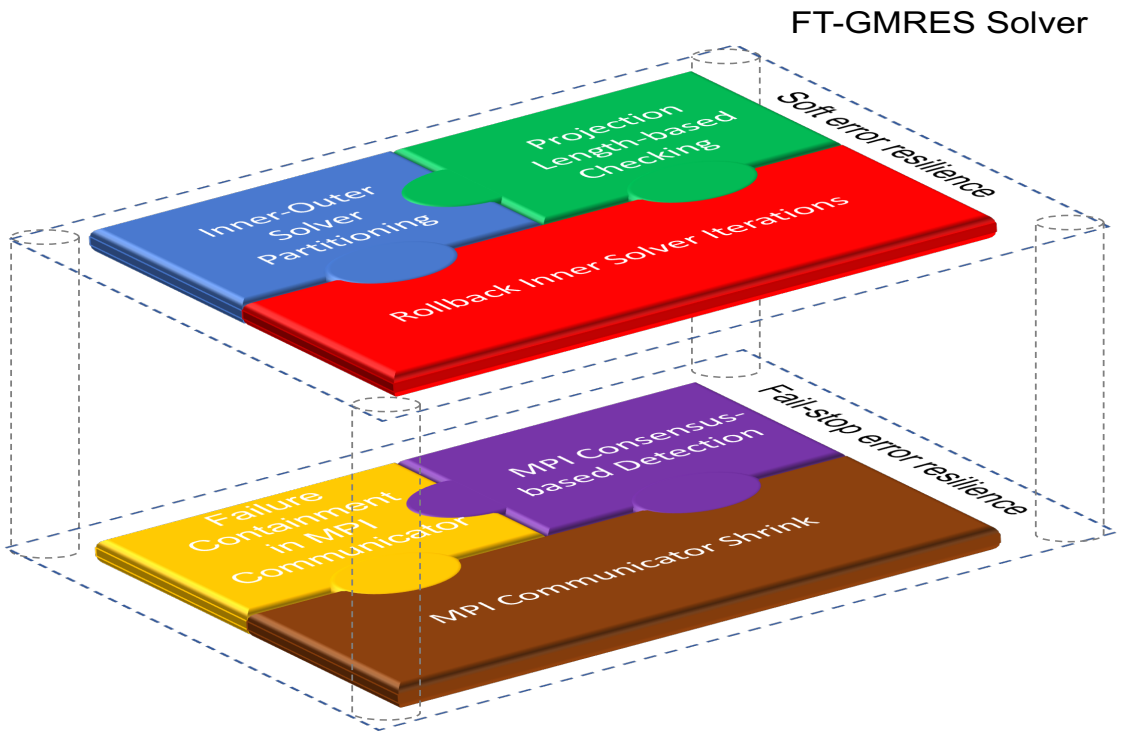


Pattern-based Modeling of Multiresilience

- Coordination among multiple patterns designed to provide optimal end-to-end application performance [3].
  - Interfaces are standardized,
  - Systematic software and hardware layer coordination.
- Navigate the performance resilience tradeoff space by evaluating multiple solutions.
- Each pattern has significantly different performance and implementation characteristics.
- Naïve stacking can lead to overprotection resulting in degradation of application performance.

Linear Solver

- GMRES generalized minimal residual method for solving non-symmetric linear systems.
  - Solve: **`Ax = b`**
  - Iterative algorithm



- Design patterns provide *detection*, *containment*, and *mitigation* for soft and fail-stop errors.

Design Patterns for Soft Error Resilience

- **State patterns:** segregation enables exploration of detection and recovery patterns, reduces overheads in most cases.
  - *Persistent state:* Matrix A and Right-hand vector b,
  - *Dynamic state:* Solution vector x,
  - *Environment state:* Data-structure indices, pointers, loop counters, etc.
- **Detection patterns:** utilize properties/characteristics of the algorithm/application/state patterns to detect presence of SDCs.
- **Mitigation patterns:** ensure forward progress of the algorithm.
  - *Compensation strategy pattern*: modular redundancy, can result in high overheads.
  - *Rollback recovery pattern*: preserve dynamic state in local memory (checkpoints).

Design Patterns for Soft Error Resilience
High Accuracy and High Cost Detector

Pattern Name	Monotonicity Violation
Problem	SDC Detection in iterative algorithms
Context	Check the progress of algorithm at each iteration by inspecting the quality metric
Forces	Applicable for iterative algorithms where **quality metric** is supposed to be **monotonically non-increasing**.
Solution	Calculate quality metric at each iteration and check violation by comparing the quality metric from previous iteration
Capability	The need to calculate quality metric frequently increases computation and communication between parallel processes.
Protection Domain	SDCs in static and dynamic state can be detected
Resulting Context	Enables timely recovery of iterative algorithm state
Rationale	Inexpensive method as compared to redundant computation

Design Patterns for Soft Error Resilience
High Accuracy and High Cost Detector

Pattern Name	Monotonicity Violation
Problem	SDC Detection in iterative algorithms
Context	
Forces	
Solution	
Capability	
Protection Domain	SDCs in static and dynamic state can be detected
Resulting Context	Enables timely recovery of iterative algorithm state
Rationale	Inexpensive method as compared to redundant computation

Quality Metric:
Residual at time step k: $r_k = b - Ax_k$
(for approximate solution x_k)

Criteria:
Monotonic decrease in quality metric
$r_k <= r_{k-1}$

Design Patterns for Soft Error Resilience
Inaccurate and Low Cost Detector

Pattern Name	Bounded Computations
Problem	SDC Detection in critical computations
Context	Check the progress and integrity of algorithm by inspecting the outputs produced during critical computations
Forces	Applicable for algorithms with identifiable **critical computations** and deterministic **lower and upper bounds**
Solution	Compare key outputs produced during critical computations against lower and/or upper bounds
Capability	Utilize implicit calculations and local invariant checking
Protection Domain	SDCs in static and dynamic state can be detected
Resulting Context	Enables timely recovery of iterative algorithm state
Rationale	Inexpensive method as compared to redundant computation

Design Patterns for Soft Error Resilience
Inaccurate and Low Cost Detector

Pattern Name	Bounded Computations
Problem	SDC Detection in critical computations
Context	tputs
Forces	d
Solution	nst lower
Capability	
Protection Domain	SDCs in static and dynamic state can be detected
Resulting Context	Enables timely recovery of iterative algorithm state
Rationale	Inexpensive method as compared to redundant computation

Critical Computations:
Inner products in each iteration of Arnoldi process (orthogonalization kernel) inside GMRES algorithm

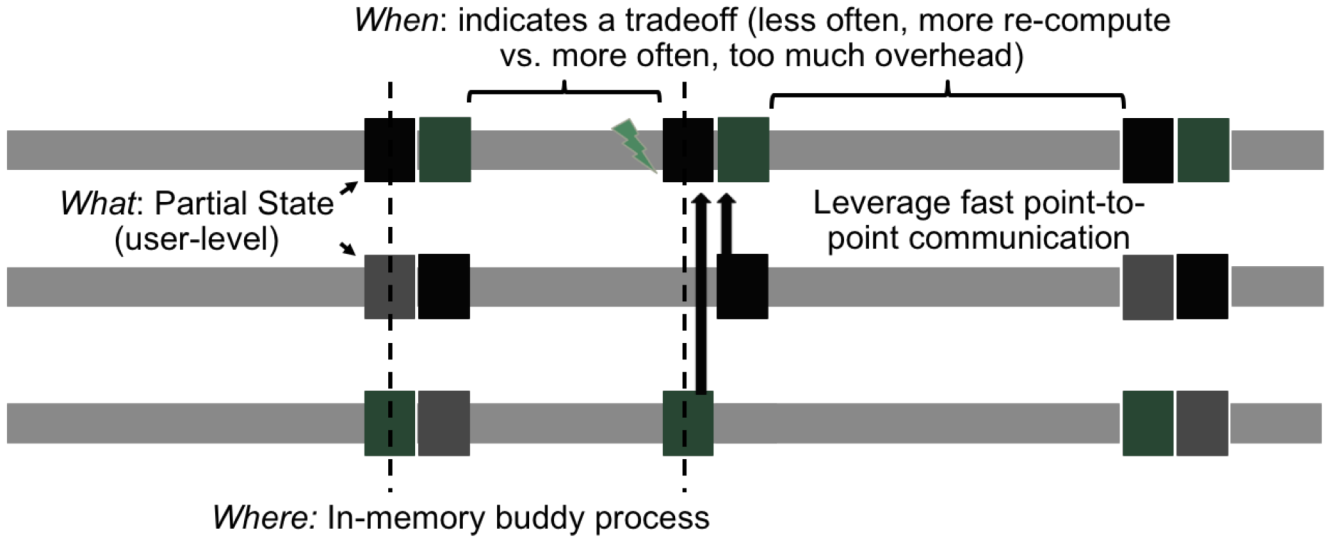
Criteria:
Theoretical upper limit for values in Hessenberg Matrix
$|h_{ij}| <= ||A||_F$,
bounded by Frobenius norm of input matrix (calculated once)

Design Patterns for Hard Error Resilience (1/2)

- **State patterns:** encapsulate the application state to facilitate recovery of lost state after process failure.
  - *Environment state:* Objects in parallel runtime environment,
  - *Persistent & Dynamic state:* Distributed across parallel processes.
- **Detection patterns:** instantiated in the environment state pattern, for robust detection and identification of failed processes.
  - *Consensus structural pattern:* proactive or reactive approach to failure detection.
- **Mitigation patterns:** recover lost persistent and dynamic state, and fix environment state for forward progress of parallel application.

Design Patterns for Hard Error Resilience (2/2)

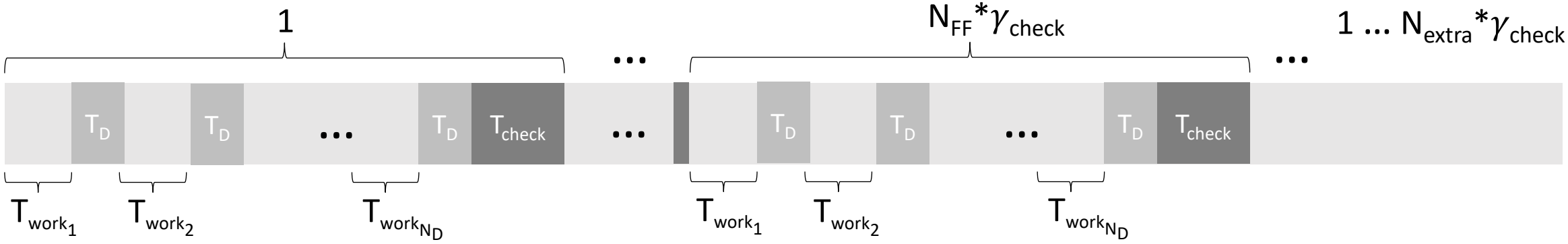
- **Mitigation patterns:** recover lost persistent and dynamic state, and fix environment state to enable forward progress of parallel application.
  - *Reconfiguration pattern:* rejuvenate parallel runtime environment by removing failed processes and refreshing parallel runtime objects for future communications.
  - *Compensation strategy pattern*: maintain a pool of spare processes for substitution.
  - *Checkpoint restart pattern:* in-memory checkpoints of persistent and dynamic state.



Checkpoint-based Multiresilience

- Utilize local checkpoints to recover from soft errors.
- In the presence of soft errors, checkpoints may become corrupted, and their use can cause the algorithm to make no progress.
- Utilize soft error detectors to verify the integrity of the checkpoints.
  - Sufficient to perform soft error detection only before performing the checkpoint.
- Performing multiple soft error detections before the checkpoint limits the propagation of soft errors and avoids costly re-computation → *fail fast*.
  - *How often should you perform the soft error detection?*
  - *Previous work [4] found optimal number of verifications to perform, but assumes ideal soft error detection. In this work, non-ideal soft error detection is assumed.*

Performance Model for Multiresilience



- Error and failure-free time for iterative algorithm:
  - Total useful work done; T_{work} is useful work done in a single iteration; N_{FF} is the total number of iterations consumed in error and failure-free condition 1
  - Overhead of performing checkpoints at a rate of γ_{check}, each time N_D soft error detections of T_D overhead each are performed 2

$$T_{FF} = \underbrace{T_{work}N_{FF}}_{1} + \underbrace{\lfloor \gamma_{check}N_{FF} \rfloor (T_D N_D + T_{check})}_{2}$$

Performance Model for Multiresilience

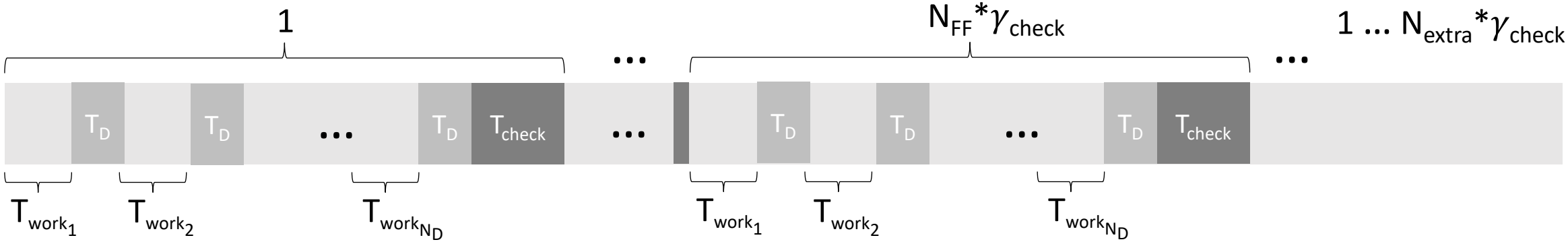
- Components of performance model:
  - Error- and failure-free total time, 1
  - Re-computation overhead due to recovery from detected soft errors ($T_{recomp\text{-}SE}$) and process failures ($T_{recomp\text{-}PF}$), 2
  - Recovery overheads from detected soft errors ($T_{SE\text{-}r}$) and process failures ($T_{PF\text{-}r}$), 2
  - Extra work beyond error free case due to presence of bounded errors or undetected soft errors; N_{extra} expected number of additional iterations beyond error free case. 3

$$T_{fail} = \underbrace{T_{FF}}_{1} + \underbrace{N_{SE}\,(T_{recomp\text{-}SE} + T_{SE\text{-}r}) + N_{PF}\,(T_{recomp\text{-}PF} + T_{PF\text{-}r})}_{2} + \underbrace{T_{work}N_{extra} + \lfloor \gamma_{check} N_{extra} \rfloor (T_D N_D + T_{check})}_{3}$$

No. of detected soft errors

No. of detected process failures

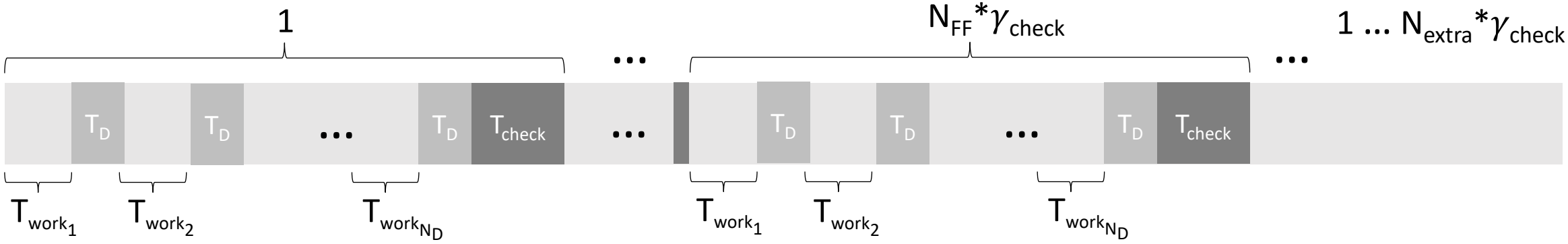
Performance Model for Multiresilience



- Re-computation Times:
  - Soft error can only be detected on the invocation of a detector, i.e., error is detected after the completion of first chunk, second chunk, and so on. 1
  - Work inside each chunk before soft error detection is done 2

$$T_{recomp\text{-}SE} = \underbrace{((1 + 2 + 3 + \ldots + N_D)/N_D)}_{1} * \underbrace{(\sum T_{work_i}/N_D + T_D)}_{2}$$

Performance Model for Multiresilience



- Re-computation Times:
  - Process failure can occur uniformly within checkpoint interval and is detected almost immediately due to its disruptive nature, given availability of runtime support.
  - Sum of all useful work done before a checkpoint is performed $\sum T_{work_i}$ 1
  - Overhead of performing multiple soft error detections (N_D) inside a checkpoint interval. 2

$$T_{recomp\text{-}PF} = (\sum T_{work_i} + T_D N_D)/2$$

1 2

Experiments

- Statistical fault injection: one of the ways to estimate parameters in the performance models.
  - N_{extra} is dependent on the characteristics of the algorithm.
- Some parameters can be calculated based on workload & system specs.
  - T_{check} depends on size of the checkpoint and latency of transfers over the network.
  - Young's formula (using MTTF) can be used to determine checkpoint frequency, γ_{check}
- Frequency (N_D) and type of error detector used affects some parameters.
  - N_{SE} number of detected soft errors, which in turn impacts N_{extra} or the overhead of additional work beyond error free case.
  - Re-computation overheads.

Experimental Goals

- Find the number of detections to perform inside a single checkpoint period for each type of soft error detector which minimizes the impact on total time-to-solution.
  - T_{fail} (N_{DH}, T_{DH}) –vs– T_{fail} (N_{DL}, T_{DL})
  - Tradeoff: overhead of detector (error-free and re-computation) –vs– extra work beyond error free case (overhead of extra checkpoints, etc.).
- Estimate N_{SE}, N_{extra} using statistical fault injection with different values of N_D
- Explore the performance resilience tradeoff space.
- Evaluate the accuracy of proposed models from estimated parameters.

Experimental Environment

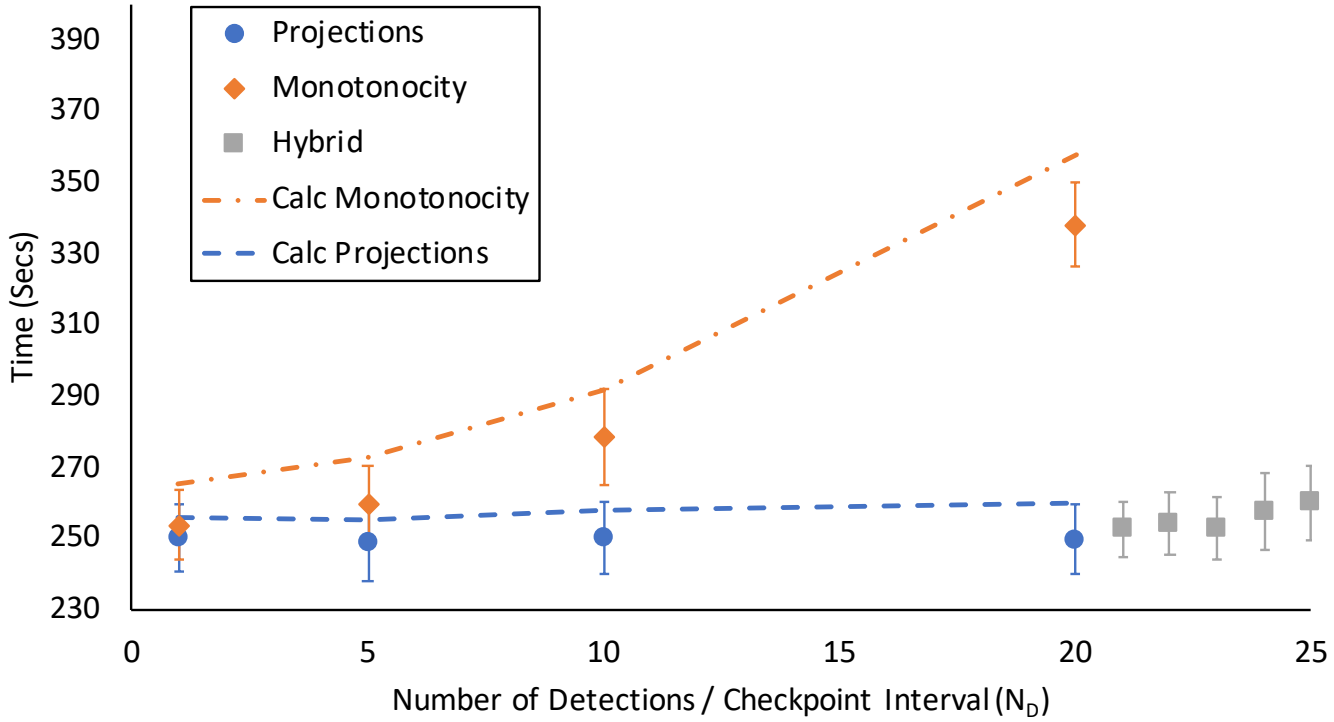
- FT-GMRES [5] implemented using Trilinos 12.6.4, https://trilinos.org/.
  - Tpetra package for parallel linear algebra using MPI.
- Parallel Environment: ULFM release 1.1, based on Open MPI 1.7.1 http://fault-tolerance.org/. User Level Failure Mitigation [6] provides:
  - Process failure detection,
  - Parallel environment reconfiguration capabilities (remove failed processes)
- Test problem: Discretization of 3D mesh. Sparse Matrix with about 7 million rows and 186 Million non-zeros.
- 40-node Linux cluster with AMD Opteron processors.
  - Cores/node: 24 (Total: 960 cores), Memory/node: 64 GB.

Experimental Setup

- Number of error and failure-free iterations, $N_{FF} = 320$.
- Checkpoint frequency constant in our experiments; checkpoint performed after every 20 iterations.
- Up to four independent process failures injected at deterministic times such that variation in re-computation (due to process failures) times is low.
- Soft errors injected randomly into computed data after almost every 10 iterations of useful work.
- Allocated max iteration count is sufficient for the solver to converge.

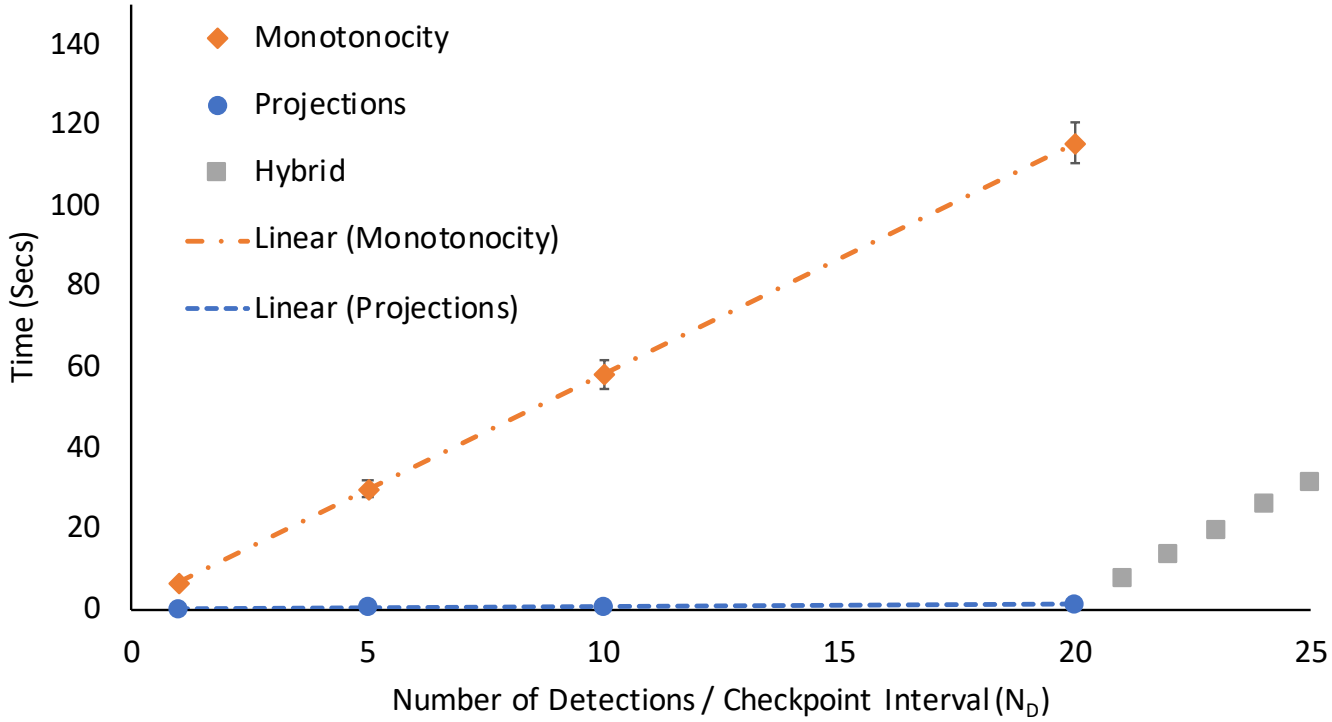
Results: Time-to-Solution

- Run-away effect of using the high accuracy and high overhead soft error detector, $N_D > 5$.
- Hybrid detector: use high accuracy detector sparingly and low accuracy detector with high frequency (1+20, 2+20, …, 5+20).



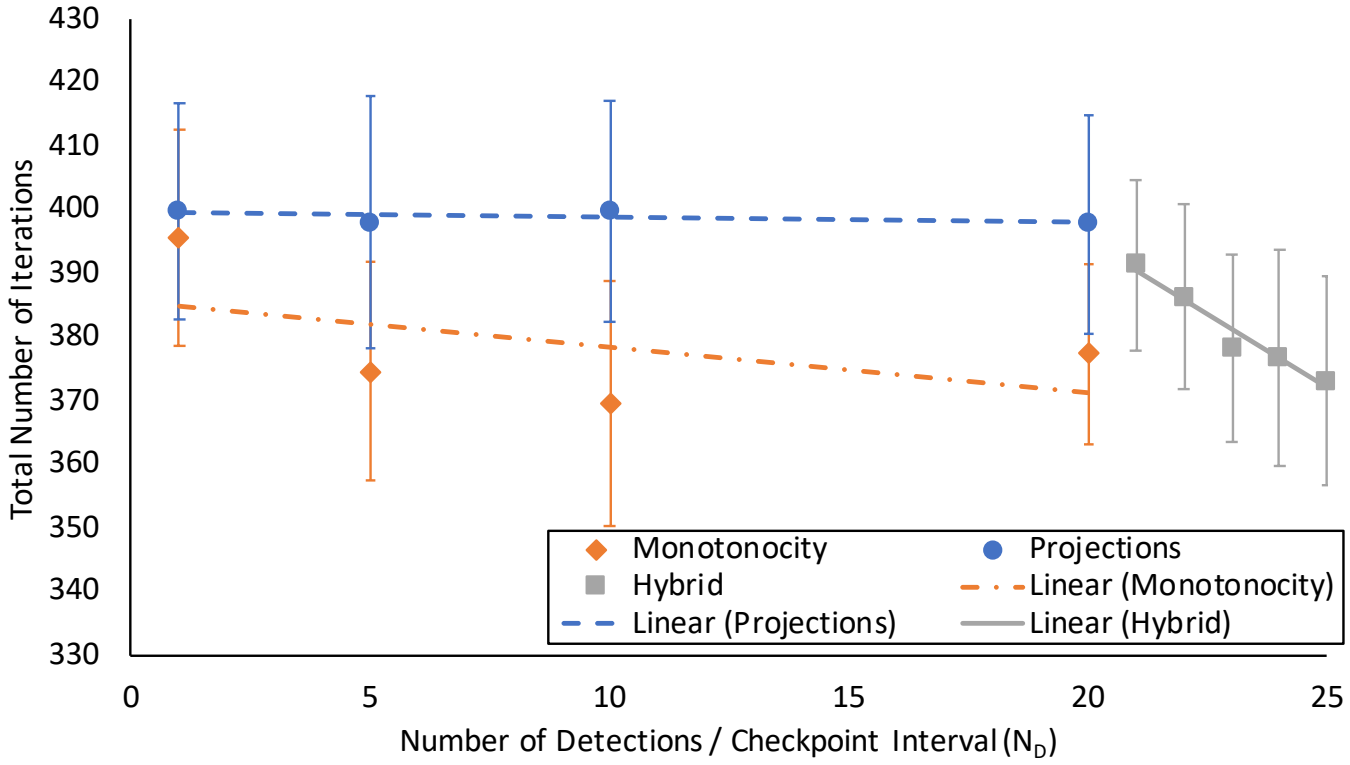
Results: Soft Error Detection Overheads

- Disparity among the overheads of the low accuracy and high accuracy detectors, $T_{DH} \cong 90\ T_{DL}$
- Using the high overhead detector with high frequency offsets any other benefits.



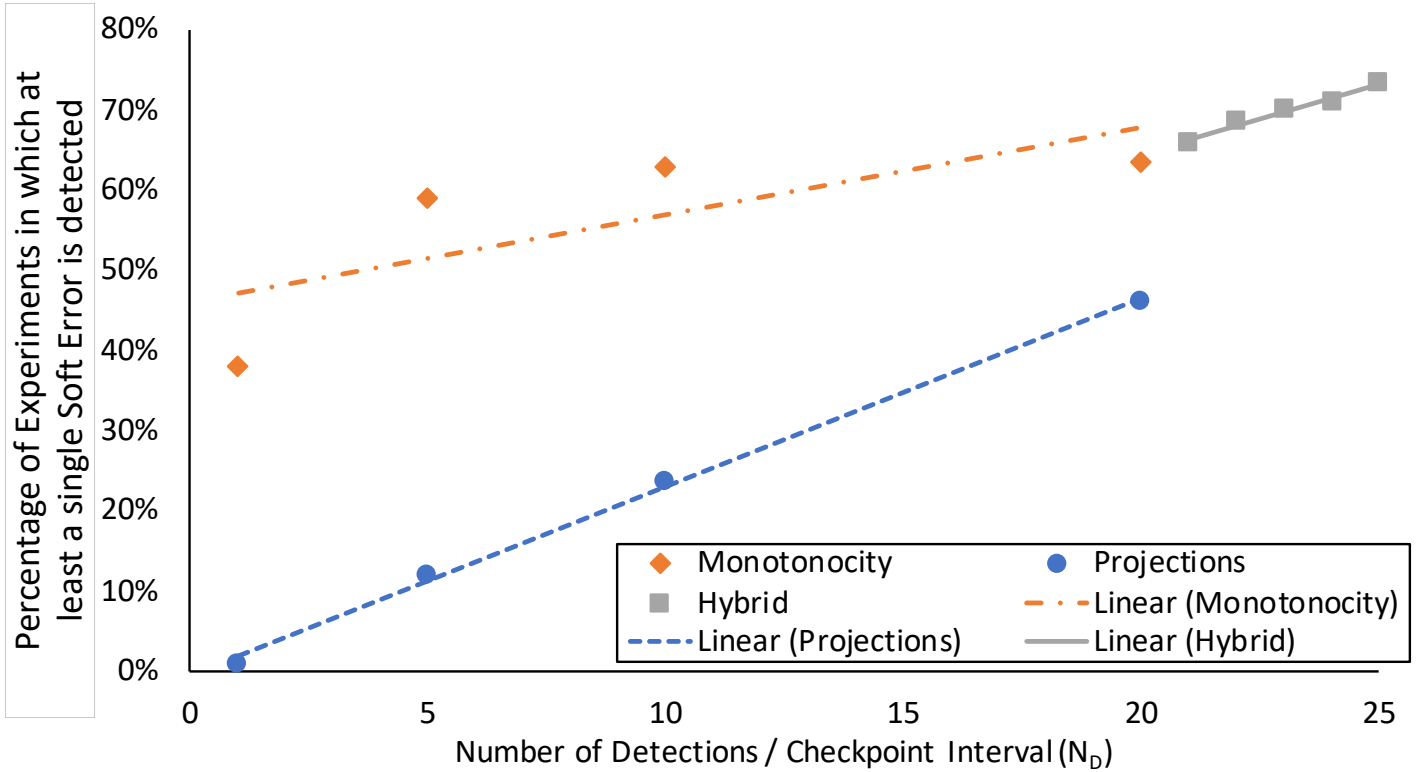
Results: Total Iteration Count

- Total iteration count includes: error- and failure-free iterations (320), re-computed iterations (detected soft errors and process failures) and additional iterations beyond error-free case.
- **Estimated N_{extra}** range **b/w 18 and 46 for high accuracy detector** and **b/w 51 and 60 for low accuracy detector** depending on number of detections performed, N_D.



Results: Detectability

- Undetected soft errors can cause any of the following: no effect on the outcome, additional work to converge to a solution, increased chances of inducing a process failure, non-convergence.
- **Estimated N_{SE}** by averaging the number of soft errors detected across all experiments for each detector. On average, **high accuracy detector caught b/w 1 and 2**, whereas, **low accuracy detector caught b/w 0 and 1** errors in each run depending on N_D.



Conclusions

- Demonstrated the design of performance efficient and multiresilient linear solver application.
- Checkpoint-based recovery in conjunction with *non-ideal soft error detection* is an effective multiresilience approach.
- Explored *performance resilience tradeoff space* using distinct soft error detectors.
- Hybrid detector is observed to have comparable detectability as using the high accuracy detector at every iteration with significantly less impact
- *Results may vary for other applications* depending on the tradeoff b/w penalty of extra work and overhead of using high accuracy detector.

References

1. Bronevetsky, G., de Supinski, B.: *Soft error vulnerability of iterative linear algebra methods.* In: 22nd Annual International Conference on Supercomputing (2008).
2. Jaulmes, L., Casas, M., Moret, M., Ayguad, E., Labarta, J., Valero, M.: *Exploiting asynchrony from exact forward recovery for DUE in iterative solvers.* In: SC15: International Conference for High Performance Computing, Networking, Storage and Analysis (2015)
3. Ashraf, R.A., Hukerikar, S., Engelmann, C.: *Pattern-based modeling of multiresilience solutions for high-performance computing.* In: Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering (2018).
4. Benoit, A., Cavelan, A., Robert, Y., Sun, H.: *Optimal resilience patterns to cope with fail-stop and silent errors*. Report RR-8786, LIP - ENS Lyon (Oct 2015).
5. Elliott, J., Hoemmen, M., Mueller, F.: *Evaluating the impact of SDC on the GMRES iterative solver.* In: IEEE 28th International Parallel and Distributed Processing Symposium. pp. 1193-1202 (May 2014)
6. Bland, W., Bouteiller, A., Herault, T., Bosilca, G., Dongarra, J.: *Post-failure recovery of MPI communication capability*. The International Journal of High Performance Computing Applications 27(3), pp. 244-254 (2013)

Contact & Acknowledgements

- Project website: https://ornlwiki.atlassian.net/wiki/spaces/RDP
- PI: Christian Engelmann, engelmannc@ornl.gov

- Work supported by the Early Career Program of U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, with program manager Lucy Nowell, under contract number DE-AC05-00OR22725.