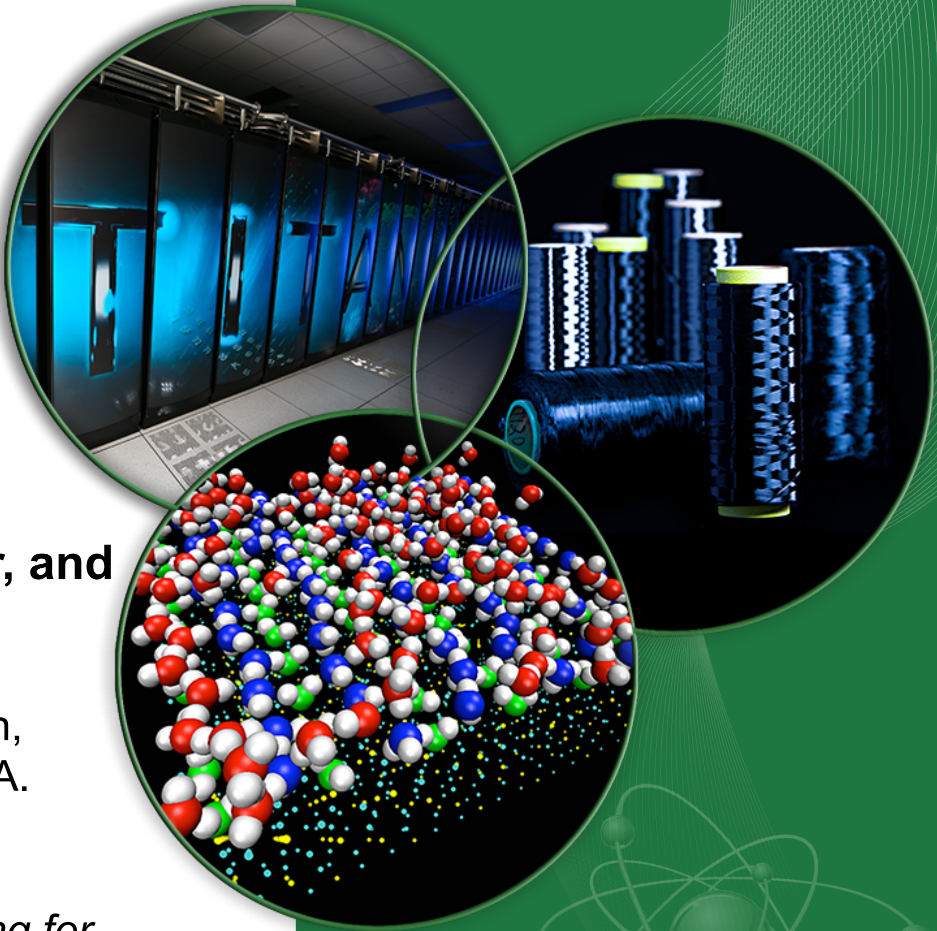
Pattern-based Modeling of Fail-stop and Soft-error Resilience for Iterative Linear Solvers



Rizwan A. Ashraf, Saurabh Hukerikar, and Christian Engelmann
Computer Science Research Group,
Computer Science and Mathematics Division,
Oak Ridge National Laboratory (ORNL), USA.

18th SIAM Conference on Parallel Processing for Scientific Computing (PP), Tokyo, Japan, March 7-10, 2018

ORNL is managed by UT-Battelle
for the US Department of Energy

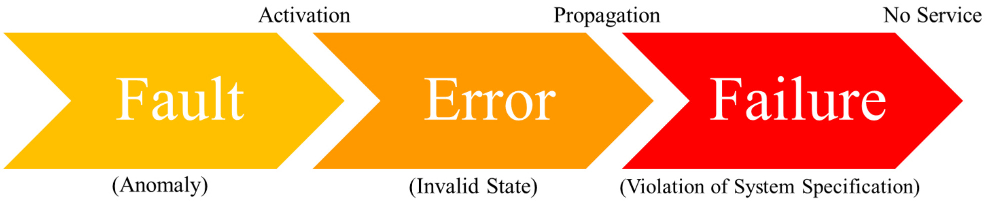
OAK RIDGE
National Laboratory

Resilience, Why?

- ***Resiliency*** in high performance computing (HPC) applications: the ability to gracefully handle errors and recover from failures.
- Errors and failures are common place in HPC systems today.
  - Large-scale systems with a number of complex & diverse software and hardware components,
  - Technology scaling trends in hardware components,
  - Complex compute, memory, interconnect and storage architectures,
  - Cost (design, area, power, engineering) of achieving error-free large scale systems is too high.
- The situation is only expected to get worse, as we move towards the goal of achieving more computational power, i.e., *Exascale systems*.

Multiresilience in HPC Applications

- HPC Applications are affected by multiple types of events which hinders with their ability to make forward progress and their correctness.
  - Soft Errors: Silent data corruptions (**SDC**),
  - Hard Errors: process failures (**Crash**),
  - Broad categorization which covers the affects of various types of faults excluding *performance faults*.
- Most works provide resilience to only a single type of error mechanism.
- Need to *systematically* integrate multiple techniques to detect and handle multiple error events, without sacrificing performance.

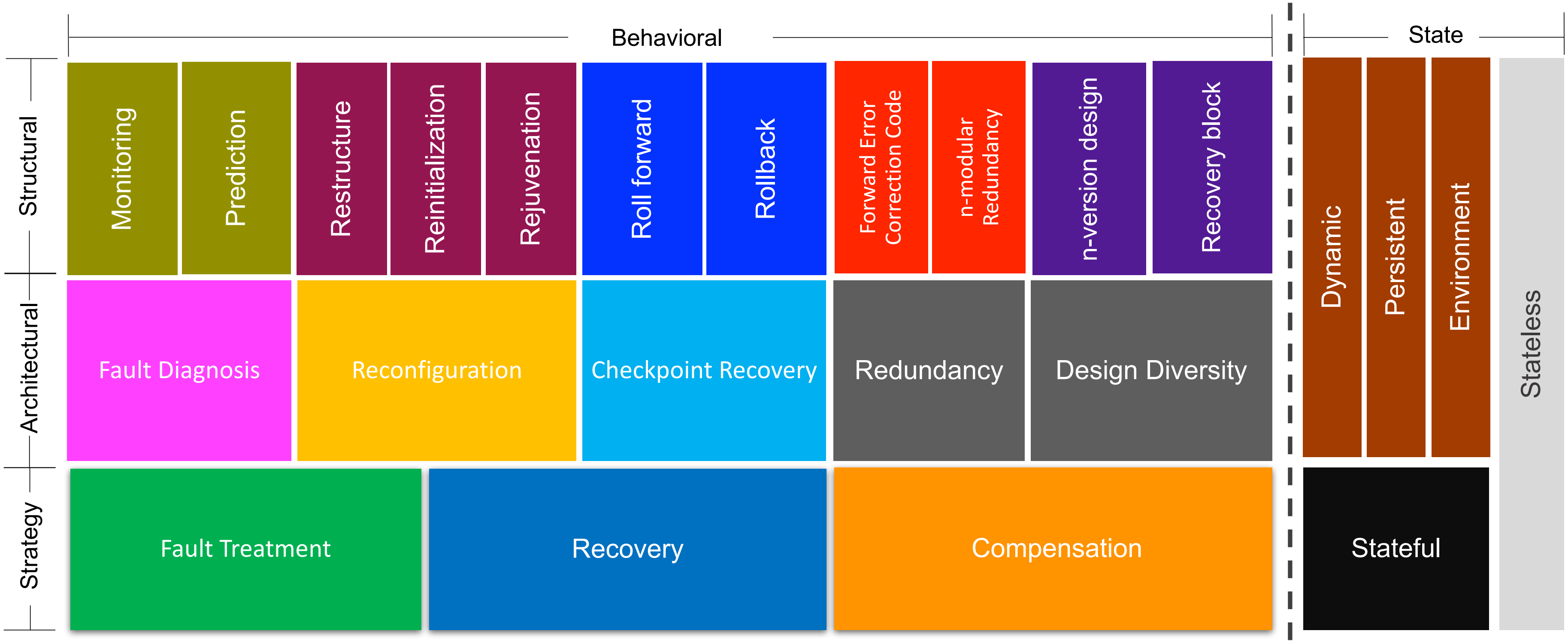


Design Patterns for Resilience

- Patterns provide a *generalizable solution* to a recurring problem.
- The solution is *formalized* with a set of activation and response interfaces, and a behavior specification.
- Patterns *do not* provide concrete solutions, instead focus on a reproducible strategy which may be used many times, implemented in different manners.
- *State Patterns*, provide encapsulation of application’s state:
  - Static/Persistent State, Dynamic State, and Environment State.
- *Behavioral Patterns,* provide detection, containment and mitigation techniques:
  - Strategy, Architecture, and Structural patterns.

Resilience Patterns Catalog

- See "*Resilience Design Patterns: A Structured Approach to Resilience at Extreme Scale*," ORNL Technical Report v1.2, December 2017.



Behavioral

State

Structural

Architectural

Strategy

Monitoring

Prediction

Restructure

Reinitialization

Rejuvenation

Roll forward

Rollback

Forward Error Correction Code

n-modular Redundancy

n-version design

Recovery block

Dynamic

Persistent

Environment

Stateless

Fault Diagnosis

Reconfiguration

Checkpoint Recovery

Redundancy

Design Diversity

Fault Treatment

Recovery

Compensation

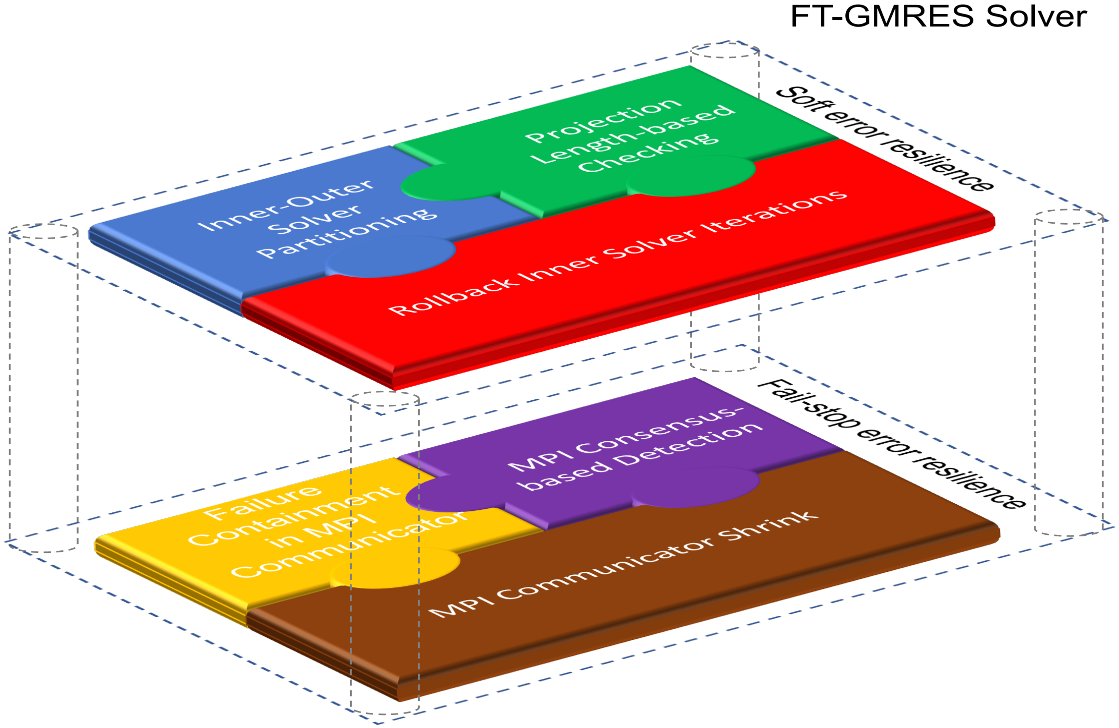
Stateful

Pattern-based Modeling of Multiresilience

- Multiple patterns are instantiated across layers of the system stack, interlinked using a *building blocks approach*.
- Coordination among multiple patterns designed to provide optimal *end-to-end application performance.*
  - 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.

Use Case: Linear Solver

- GMRES minimal residual method for solving non-symmetric linear systems.
  - Solve: `Ax = b`
  - Iterative algorithm
- Resilience patterns provide *detection*, *containment*, and *mitigation* for soft and fail-stop errors.


FT-GMRES Solver
Soft error resilience
Projection Length-based Checking
Inner-Outer Solver Partitioning
Rollback Inner Solver Iterations
Fail-stop error resilience
MPI Consensus-based Detection
Failure Containment in MPI Communicator
MPI Communicator Shrink

Patterns for Soft Error Resilience

- SDCs can cause unbounded numerical errors, which propagate across parallel processes, resulting in slow convergence of the solver.
- **State patterns:** segregation enables exploration of detection and recovery patterns, reduces overheads in most cases.
  - *Static 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.
  - *Monitoring pattern:* checks progress by keeping track of the *quality metric*. Depending on the application, the overhead of calculating the quality metric can vary widely.
  - Application specific instantiation of the monitoring pattern: *bounded-compute pattern*, verifies outputs produced during critical computations against a-priori known bounds.
- **Mitigation patterns:** ensures forward progress of the algorithm and application.
  - *Compensation strategy pattern*: modular redundancy, results in high overheads.
  - *Rollback recovery pattern*: preserve dynamic state in local memory (checkpoints).

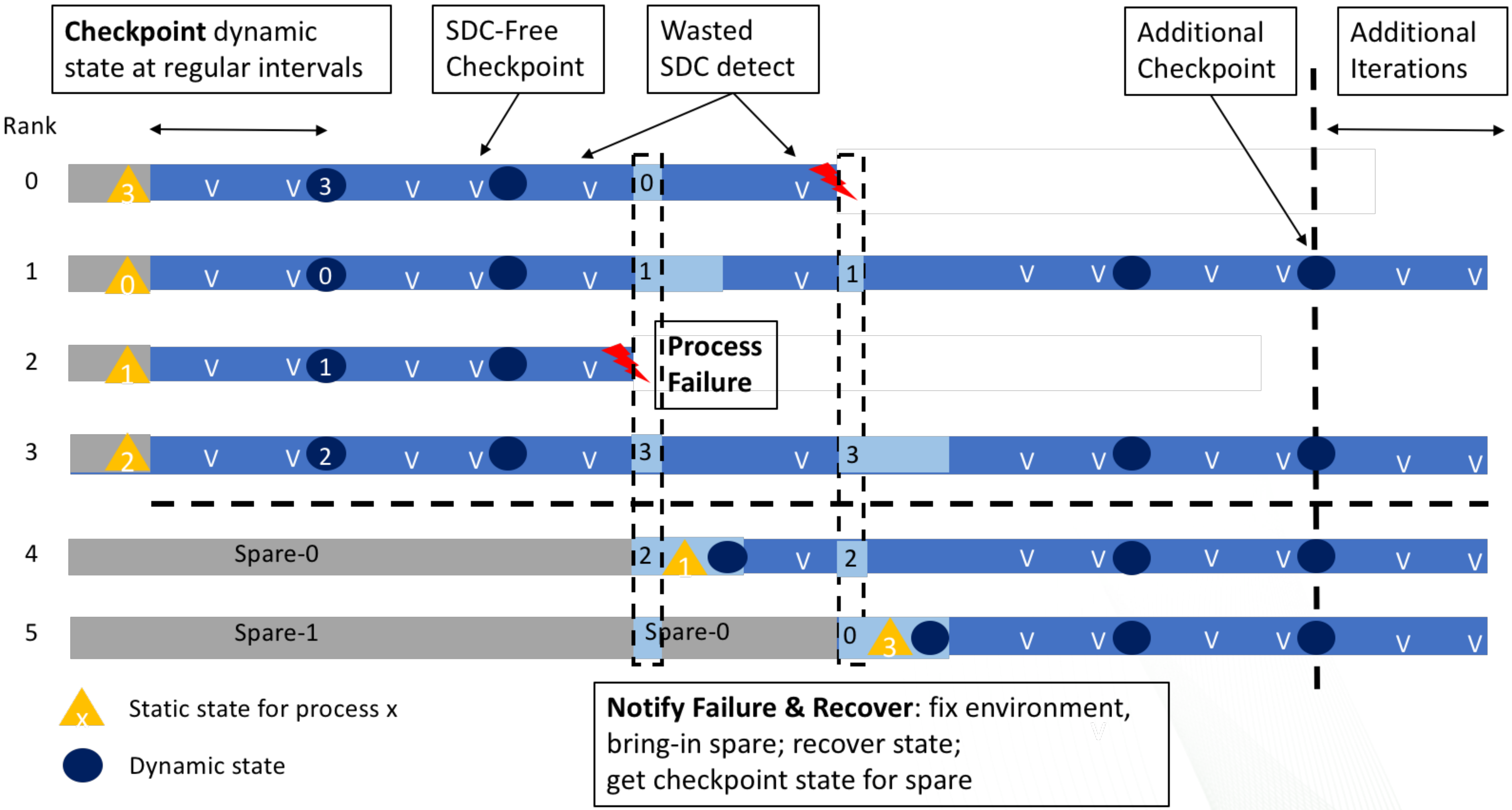
Patterns for Hard Error Resilience

- Process failures make a parallel application to stall indefinitely or result in fatal crash, within the distributed memory model, losing all work done.
- **State patterns:** encapsulate the application state to facilitate recovery of lost state after process failure.
  - *Environment state:* Objects in parallel runtime environment,
  - *Static & 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.
  - Proactive strategy: Collective operations (all process communication) can be strategically placed to enable runtime environment to detect failures regularly.
- **Mitigation patterns:** recover lost static and dynamic state, and mitigate environment state for 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 replacement.
  - *Checkpoint restart pattern:* remote in-memory checkpoints of static and dynamic state.

Pattern-based Modeling

Fault Model	Pattern Class	Choices	Selection
Soft Error	State	Dynamic, static, environment	Need scoping of *dynamic and static state* only, since environment corruption results in process failure.
	Detection	Monotonicity, bounded compute, checksums	*Bounded compute* results in 14x lower overhead compared to monotonicity monitoring pattern.
	Recovery	Checkpoint, checksums	*Local in-memory checkpoints* due to less computational overhead.
Hard Error	State	Dynamic, static, environment	*All*, since process failures are fatal for application.
	Detection	Proactive, reactive	*Proactive*, since collectives present in every iteration, prevent propagation.
	Recovery	Environment: warm spares, rejuvenate	*Spares* to avoid re-allocation of state.
		Checkpoint-restart, diskless checkpoints, linear interpolation	*In-memory checkpoint-restart* because of least overhead and no effect on convergence of solver.

Multiresilience - Pattern coordination



Checkpoint dynamic state at regular intervals

SDC-Free Checkpoint

Wasted SDC detect

Additional Checkpoint

Additional Iterations

Process Failure

Notify Failure & Recover: fix environment, bring-in spare; recover state; get checkpoint state for spare

Static state for process x

Dynamic state

Experimental Setup

- FT-GMRES implemented using Trilinos 12.6.4 framework, 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/. ULFM provides:
  - Process failure detection,
  - Parallel environment reconfiguration capabilities (remove failed process)
- 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,
  - Point-to-point bandwidth: 215 MB/s.

Goals of Experiments

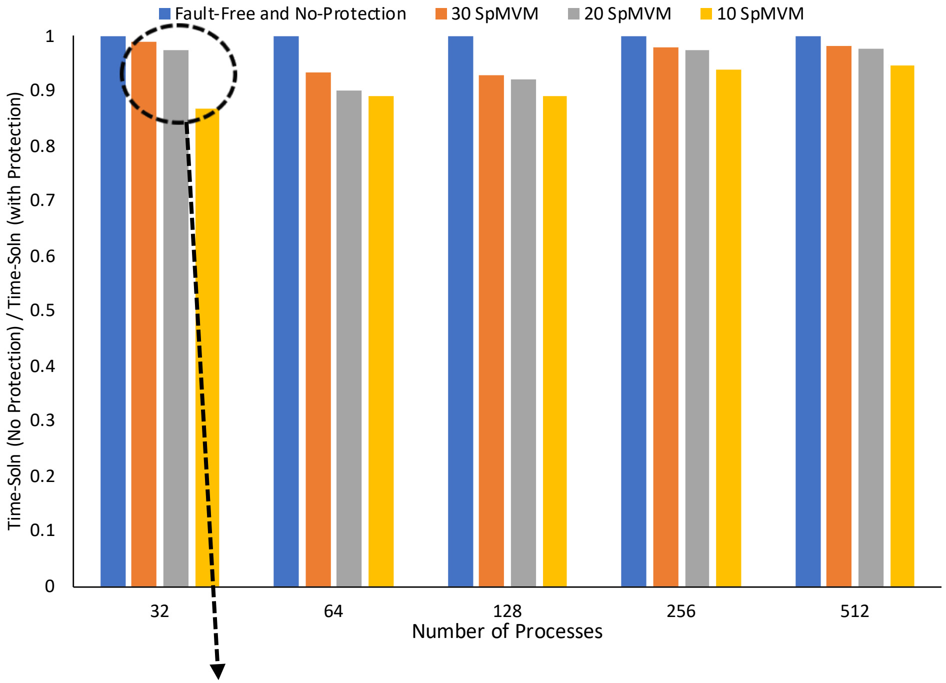
- **Goal # 1:** Evaluate the resiliency and performance characteristics of individual patterns.
  - Design reproducible error injection experiments, e.g., coefficient of variation for all experiments ranges between 0.01 and 0.05.

- **Goal # 2:** Quantify the performance interactions between soft error and process failure resilience patterns in a multiresilience solution.
  - Identify constraints of combining patterns.

Results – Soft Error Resilience

- Soft Error injected after every 30 (less frequent), 20 and 10 (more frequent) Sparse Matrix Vector Multiplication (SpMVM) operations.
- In all cases, the solver converged to a correct solution in allotted time.

Higher is better (less overheads)

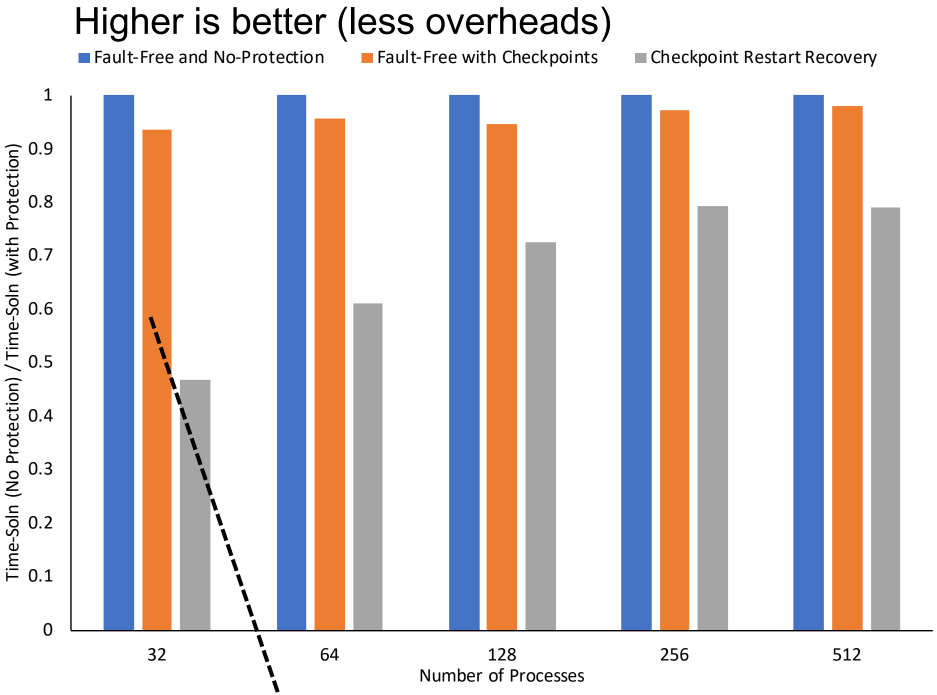


Overheads: Detection, recovery, additional iterations.
Tradeoff: High detection overhead, less additional iterations.

Processes	Detect + Recovery Overheads		Num. of Additional Iterations	
	30 SpMVM	10 SpMVM	30 SpMVM	10 SpMVM
32	2.1%	8.9%	30 [75]	36 [150]
64	6.5%	4.8%	27 [50]	34 [75]
128	8.1%	8.8%	25 [25]	33 [125]
256	1.1%	1.9%	32 [50]	36 [125]
512	0.7%	0.7%	28 [50]	35 [100]

Results – Process Failure Resilience

- Processes terminated based on exponential distribution with constant failure rate of time to complete 75 iterations. Up to four independent process failures.
- Pre-selected processes for termination gives results with low standard deviation.

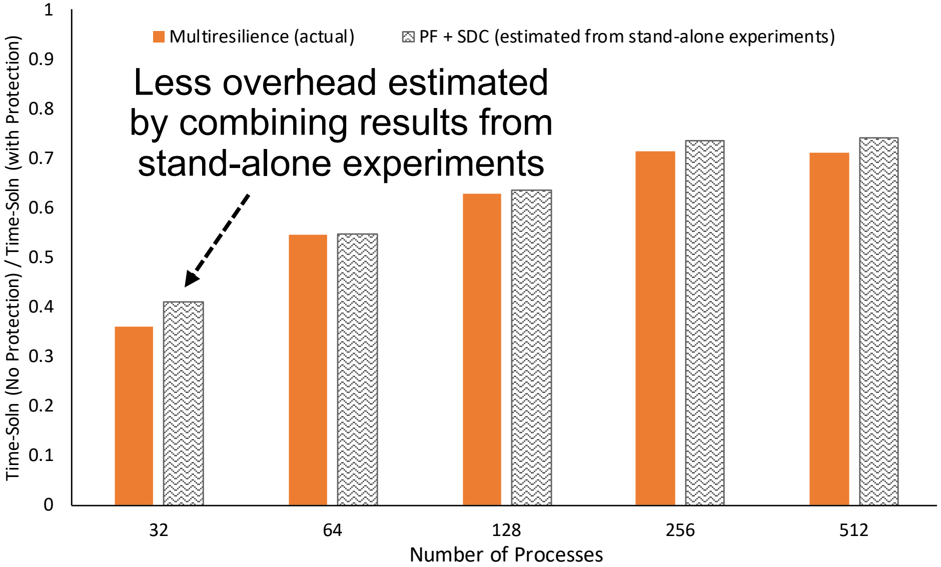


Overheads: **[No fail:]** Checkpoint of static (once) and dynamic states; **[Failures:]** No fail + state recovery, recompute, checkpoint (static, dynamic) state spares.

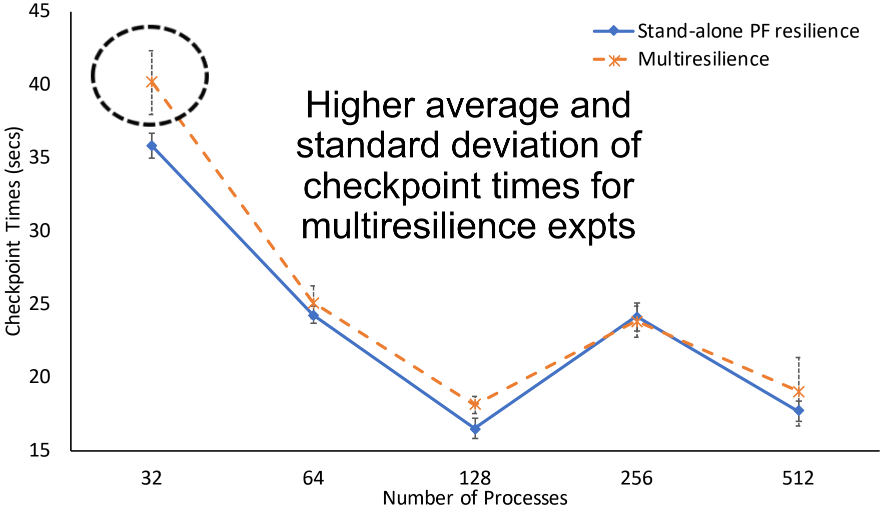
Processes	**Reconfig**	**Recover State**	**Checkpoint Static + Dynamic [%dynamic]**	**Re-compute Overhead**
32	0.02%	17.1%	28.1% [25.6%]	10.9%
64	0.03%	9.4%	18.5% [22.9%]	13.4%
128	0.04%	5.4%	12.9% [14.7%]	12.9%
256	0.02%	1.9%	7.5% [16.7%]	13.5%
512	0.05%	1.2%	5.1% [12.2%]	16.2%

Results – Multiresilience

- Multiresilience to soft errors injected after every 10 SpMVMs (multiple times in a checkpoint interval) and up to four process failures.
- More overhead of dynamic state checkpoints due to increase in convergence time as compared to stand-alone process failure experiments.
  - On average, additional time is less than expected failure time, otherwise difference will be significant since checkpoint of static state is expensive.
- Overhead of wasted soft error detections is negligible in our experiments.



Less overhead estimated by combining results from stand-alone experiments



Higher average and standard deviation of checkpoint times for multiresilience expts

Conclusions

- A *pattern-oriented design and implementation* approach for gracefully handling multiple error modes.
- Iterative refinement of pattern relationships to *optimize end-to-end application performance*.
- A generalizable approach to *architect resilience or multiresilience solutions* by composing patterns from multiple layers of system stack.
- *Experimental evaluation* for an iterative linear solver application exploiting algorithmic patterns.
- Highlighted the importance of considering *interactions between patterns* when designing a multiresilience solution.