Resilience Assessment and Enhancement

SUPER — Institute for Sustained Performance, Energy, and Resilience

Chunhua Liao, Bronis R. de Supinski, Todd Gamblin, Kathryn Mohror, Dan Quinlan
Lawrence Livermore Natl. Lab.

Prasanna Balaprakash, Paul Hovland, Stefan Wild,
Argonne Natl. Lab.

Saurabh Hukerikar, Pedro Diniz, Bob Lucas
USC-ISI

George Bosilca, Thomas Herault
University of Tennesse

Ganesh Gopalakrishnan
Univ. of Utah
with contributions from: Greg Bronevetsky (Google Inc.)

Robert J. Fowler, Anirban Mandal
RENCI

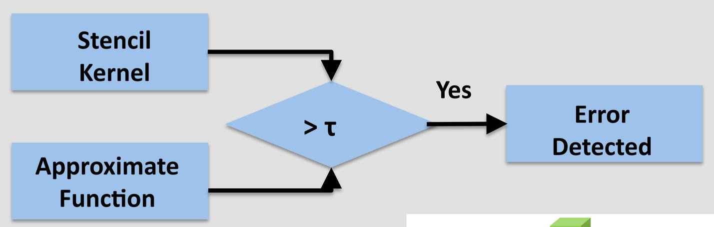
David Lowenthal
The University of Arizona

U.S. Department of Energy — Office of Science

Error Detector Synthesis

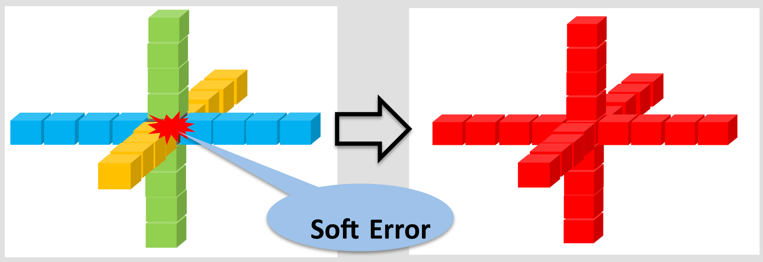
Automated Error Detector Synthesis via Machine Learning

- Protect stencil computations against soft errors
- Stencil based computations have high arithmetic intensity
- Target PDE solvers using stencil based computations
- Main memory and data cache often protected using ECC
- Focus on soft errors affecting CPU registers and its ALU
- Key idea is to use a variant of software level DMR
- Use approximate function for redundant computing
- Approximate function derived using machine learning



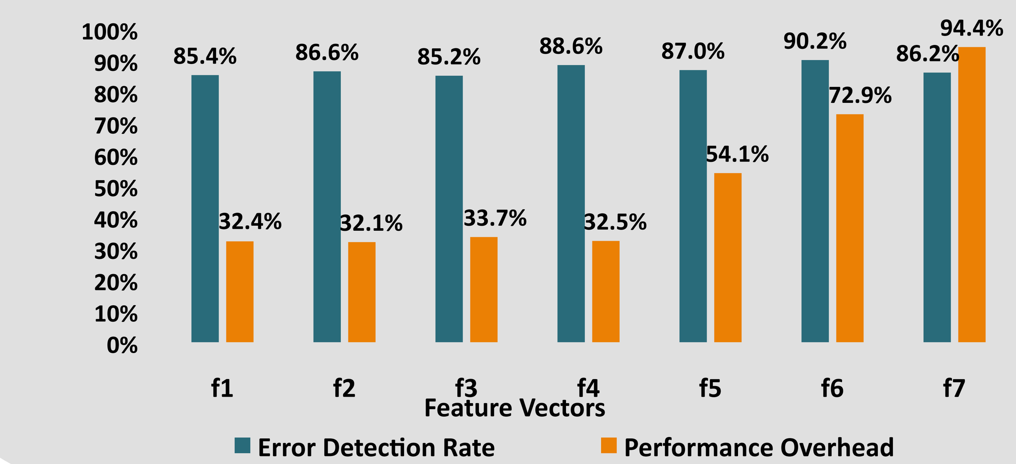
Outcome, Tools:

- Ideas implemented and released in tool Sorrel www.cs.utah.edu/fv/FMR
- Found effective at trapping soft errors
- Overhead mitigation techniques being studied



time step t — time step t+1

Soft-error Propagation in 25-point RTM Stencil



Programming Model for Resilience

Amelioration of Errors in Data Structures

- re-initialization of key variable and either roll-back or roll-forward
- user-provided amelioration function
- redundant information in the form of checksums for matrices

```
#pragma resilience recover-rollback
reinitialize (variable_list) {
  <code block>
}
```

```
#pragma resilience recover-rollforward
reinitialize (variable_list) {
  <code block>
}
```

```
#pragma resilience recover-rollback
ameliorate (recovery_func) {
  <code block>
}
```

```
#pragma resilience recover-rollforward
ameliorate (recovery_func) {
  <code block>
}
```

- association of amelioration function at allocation

```
void *resilience_malloc_repairable (...,
(void)(heal_func())*);
```

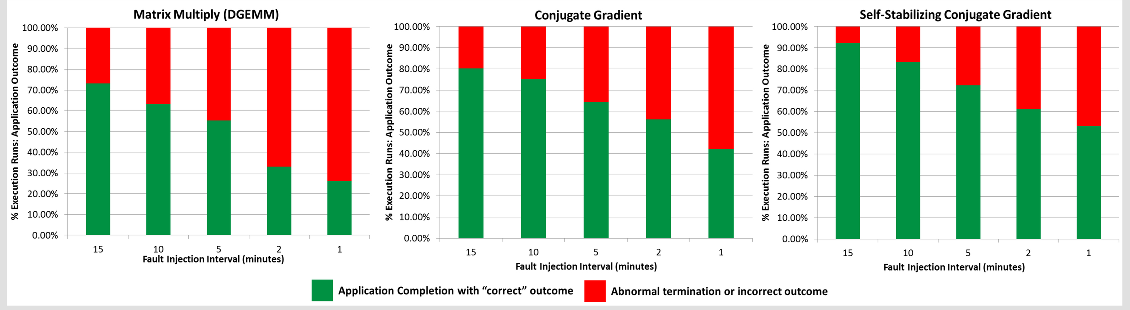
```
heal(recovery_func()) float* matrix[N][N];
```

Experiments

- Sequential Executions
- Accelerated Fault Injections
  - Errors in text and code segments
  - 1 fault every 15/10/5/2/1 mins.
  - Long application runs: 20 mins. Min.

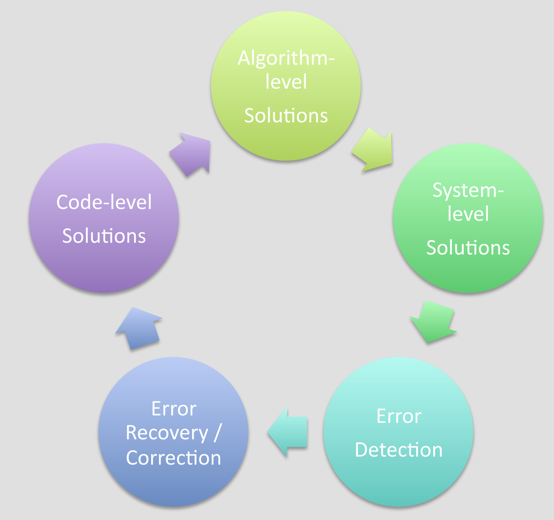
Codes:

- **Matrix-Matrix Multiplication** – uses row and column checksums for operand matrices A and B and the resilient malloc directive;
- **Conjugate Gradient Solver** – uses checksum in matrix A and protecting CG iterations steps with roll-forward using the checksum to validate correctness of the operand matrix;
- **Self-Stabilizing Conjugate Gradient** – protects iteration step using roll-backward and user-provided stability function for amelioration.



Abstract

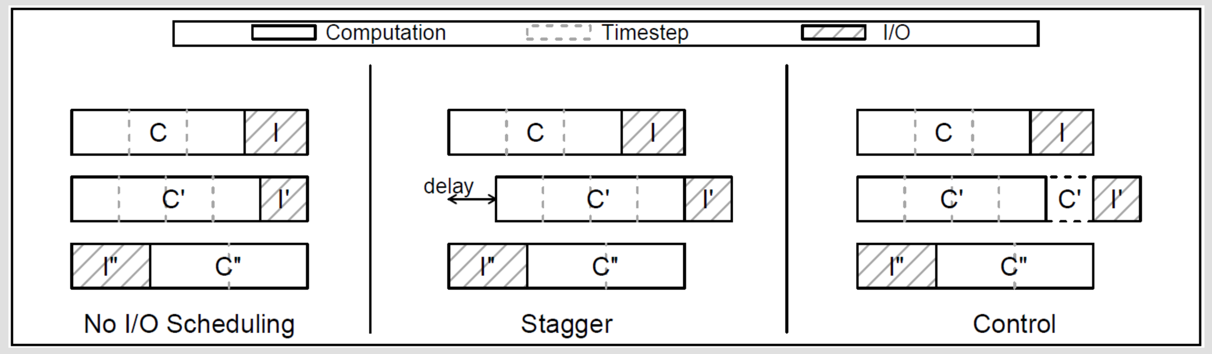
We are addressing the problems in software resilience with a holistic multifaceted approach that spans across software levels. One approach emphasizes tracking expected control flows or data invariants, and is aimed at detecting silent data corruption. Another explores language extensions and compiler technology to convey to compilers and run-time system resilience properties of code sections and algorithms. Additionally, we are investigating specific algorithmic properties of applications to develop fault tolerant extensions to dense and sparse methods. At the highest levels, we detect silent-data corruption by replicating and comparing values across MPI processes and improve on the state of the art for checkpoint/restart with innovations in file systems and checkpoint compression.



I/O Aware Power Shifting

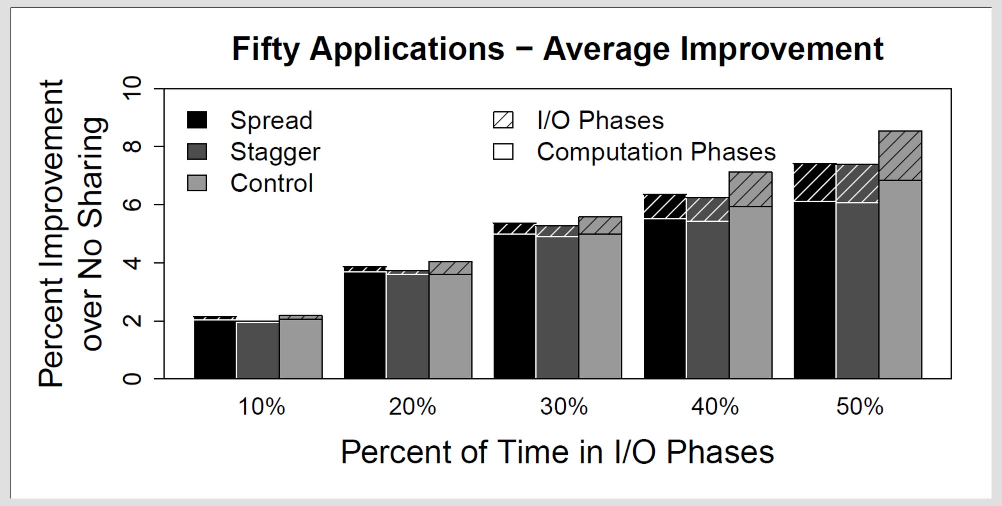
Background

- Future HPC systems will likely be power constrained.
- Many applications include periodic, low-power I/O phases.
- Thus, applications may be power limited during computation phases but have extra power during I/O phases.
- This extra power can be reallocated to other applications to improve performance.
- The performance improvement will be greatest if we minimize overlap between I/O phases of different applications.



Algorithms

- **Stagger:** attempt to avoid overlap of I/O phases by delaying some applications by a small amount.
- **Control:** I/O phase times are controlled centrally. I/O phase overlap is reduced by allowing some applications to perform extra computation iterations before entering an I/O phase.
- **Spread:** No attempt is made to prevent I/O phase overlap; instead, power is shifted on a best-effort basis.

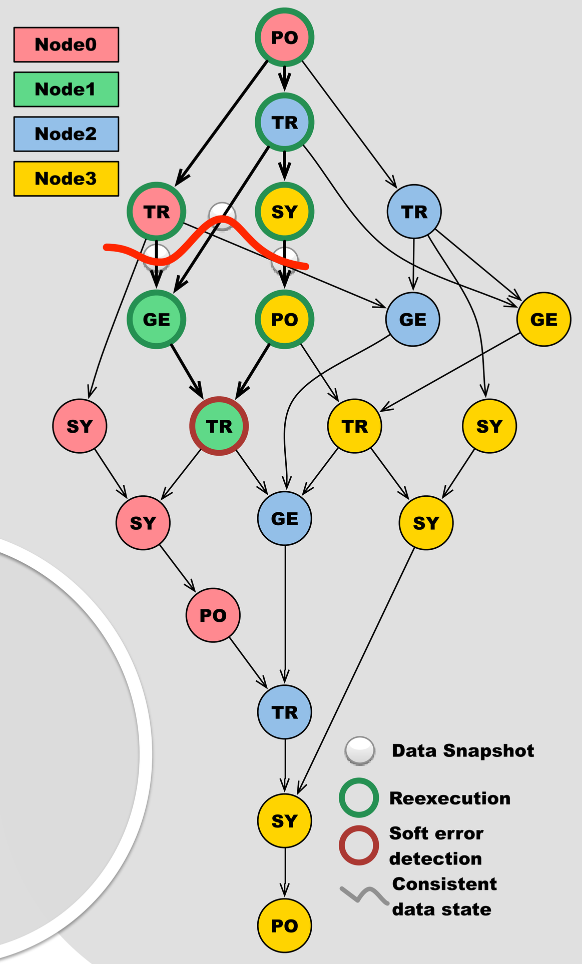


Results

- Performance improvements range from 2% when I/O phases take 10% of execution time to 8% when I/O phases take 50% of execution time.
- No applications experience performance degradation.

Resilient Task-based Run-Time

PaRSEC: generic runtime for architecture-aware scheduling of micro-tasks on distributed many-core heterogeneous architectures



Concepts

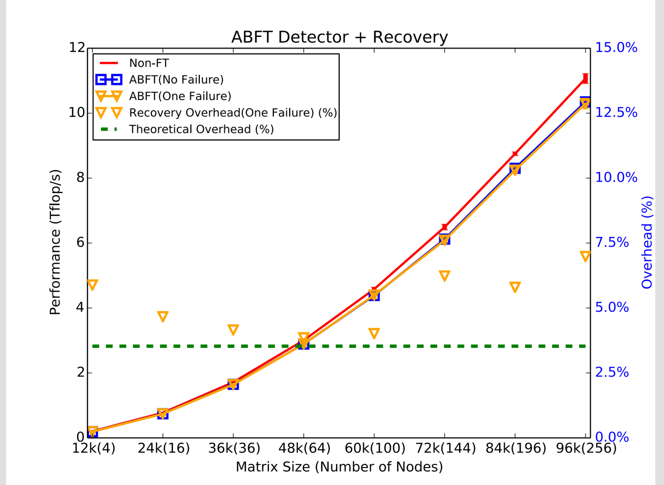
- Clear separation of concerns: compiler optimize each tasks, developer describe dependencies between tasks, the runtime orchestrate the dynamic execution
- Interface with the application developers through specialized Domain Specific Languages (PTG, insert_task, fork/join, …)
- Separate algorithms from data distribution
- Expose maximal parallelism by minimizing the control flow

Runtime

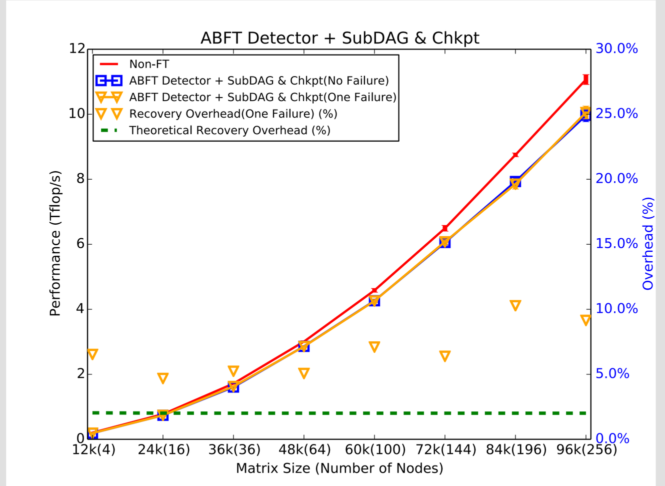
- Permeable portability layer for heterogeneous architectures
- Scheduling policies adapt every execution to the hardware & ongoing system status
- Data movements between consumers are inferred from dependencies. Communications/computations overlap naturally unfold
- Coherency protocols minimize data movements
- Memory hierarchies (including NVRAM and disk) integral part of the scheduling decisions

Resilience

- Data versioning, copy-on-write, data logging tracks changes applied on the data.
- Variable interval data logging (snapshot) based on algorithm properties, accepted overhead, amount of extra memory, and hardware MTBF.
- Whenever a task fails the validation stage, or the OS inform the runtime about unrecoverable memory corruptions, the runtime can automatically build a minimum spanning recovery DAG composed of all paths from snapshot data to the failed task.
- The recovery DAG is then executed in parallel with original application DAG, minimizing the overhead
- Snapshot based methods are generic and provided automatically by the runtime. ABFT methods require data validators provided by the algorithm developer
- Application developed on PaRSEC are resilient.



Algorithm Based Fault Tolerance checksums are used to maintain consistently valid data during the execution.



Instead of ABFT-based recovery, the runtime keeps copies of older versions of the data in order to minimize the need for re-execution (checkpoint interval once every 10 updates).

Modeling and Optimization

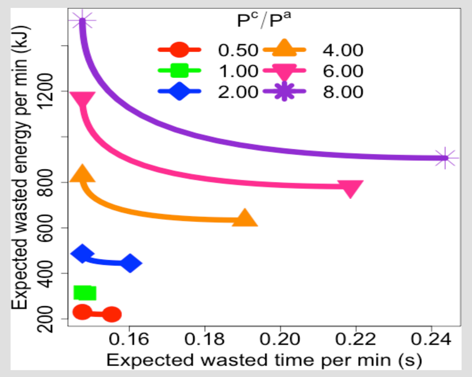
Objectives

- Develop analytical models for both expected run time and energy consumption for multilevel FTI checkpointing schemes under generic error rates
- Characterize the Pareto-optimal solution set based on varying checkpoint frequency and investigate the tradeoffs between expected time and energy consumption
- Perform power consumption measurements of large-scale executions on an IBM Blue Gene/Q with several applications
- Experimental study to analyze several system-level parameters (such as I/O power) for multilevel checkpointing that can potentially impact the tradeoffs

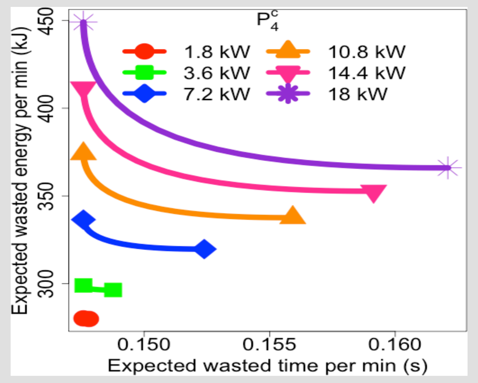
Analytical Modeling

time (energy) for a failure-free execution of an application
+
expected time (energy) wasted due to failures and checkpointing **(copy, rework, restart, downtime)**

Impact of C/R power



Impact of I/O power



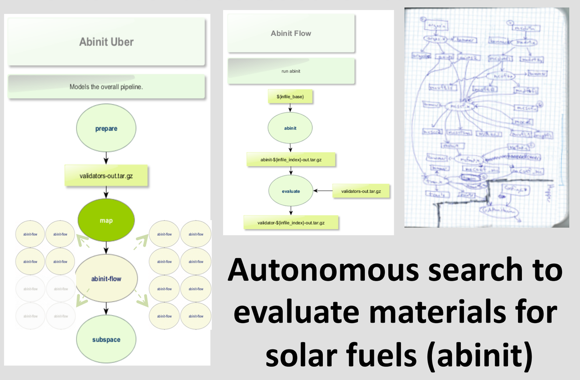
Conclusions

- I/O power has a significant impact on tradeoffs
- Increases in power for computation increases energy consumption, but yields minor tradeoffs
- Power for computation should be significantly less than that for checkpointing in order for richer tradeoffs to exist
- ***Analysis of the tradeoffs between energy and run time for multilevel checkpointing***. Balaprakash, Gomez, Bouguerra, Wild, Cappello, and Hovland. ***PMBS 2014***.
- ***Energy-performance tradeoffs in multilevel checkpoint strategies***. Balaprakash, Gomez, Bouguerra, Wild, Cappello, and Hovland. ***Cluster 2014***.

Resilient Workflows

Objectives

- An increasing fraction of high-end workloads are (dynamic) ensembles of workflows, e.g. UQ campaigns
- Failures occur across hardware, software, and application/model layers
- These workloads must be run autonomously in unreliable environments
- Capacity runs need orchestration to "run past faults"
- Many need to be co-located in high-end centers because of data communication requirements.
- Autonomous management of computational campaigns
- Develop support for multi-domain, network-aware policies for data-movement when ensemble and workflow members cross domain boundaries



Autonomous search to evaluate materials for solar fuels (abinit)

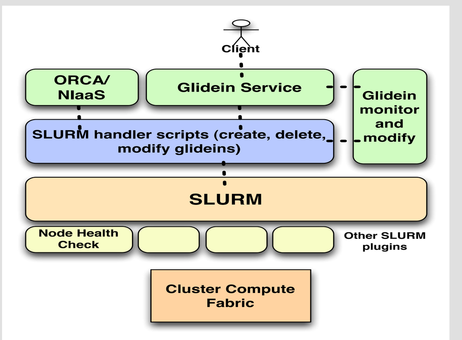
Application Engagement

- Engaging with DAKOTA toolkit (Sandia Labs)
- Studied different parallelism use cases for the DAKOTA toolkit for running ensembles
- Did preliminary performance analysis of the DAKOTA toolkit
- Resilient glideIns to be integrated with the "Job tiling" parallel use case using **SLURM**

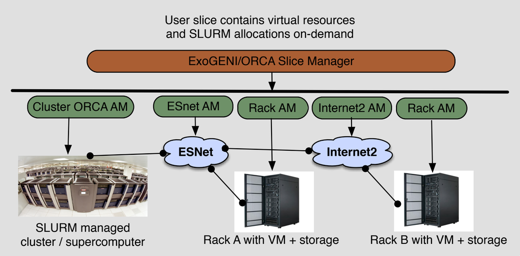
Approach

- Ensemble management using glideins
- Create tool for **SLURM** to create, delete, and modify glideIns of pre-defined types
- GlideIns should be transparently resilient to node failures by leveraging and extending health-check plugins
- Capability to extend glideIns to virtual networked environments (out-bursting) and vice versa (in-bursting) for seamless data movement

SLURM glideIns



Cross-domain



(Partial) support for this work was provided through Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (and Basic Energy Sciences/Biological and Environmental Research/High Energy Physics/Fusion Energy Sciences/Nuclear Physics). LLNL-POST-657319.

Argonne National Laboratory — USC School of Engineering, University of Southern California — Innovative Computing Laboratory, The University of Tennessee — The University of Utah — renci — The University of Arizona

(Partial) support for this work was provided through Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (and Basic Energy Sciences/Biological and Environmental Research/High Energy Physics/Fusion Energy Sciences/Nuclear Physics) under award numbers DE-SC0006844 and DESC0006947.