Building a Resilience Autotuning Framework: SUPER's integrated resilience strategy

Bronis R. de Supinski, Marc Casas-Guix, Dan Quinlan, Chunhua Liao
LLNL

Pedro C. Diniz, Bob Lucas
USC-ISI

Ganesh Gopalakrishnan
University of Utah
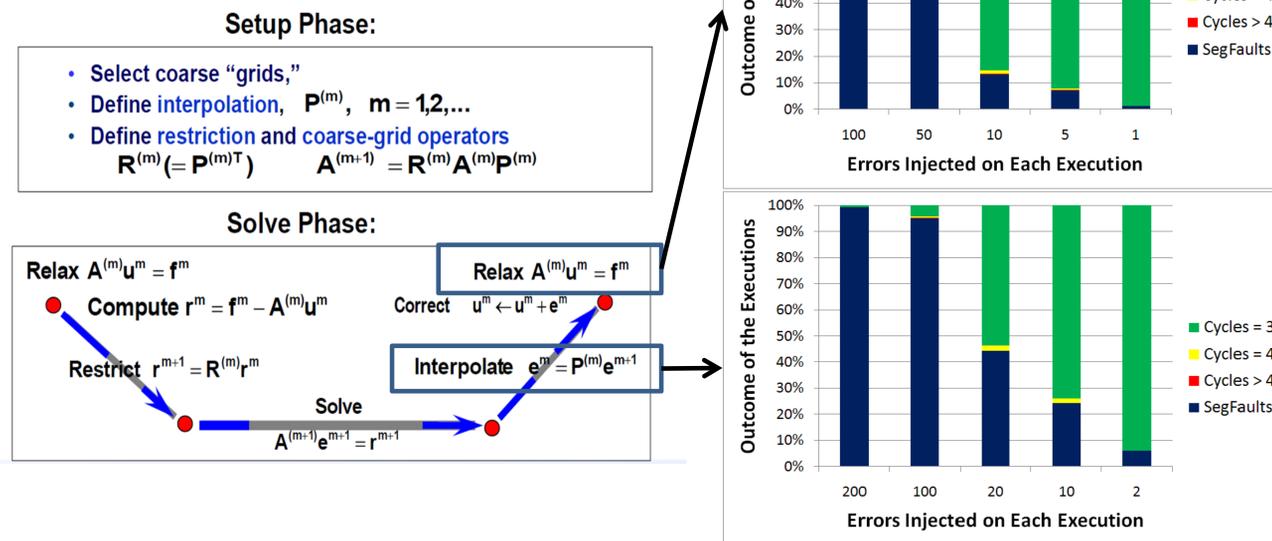
George Bosilca
University of Tennesse

SUPER is creating a comprehensive tool set to evaluate application fault vulnerability, to reduce it and to assess trade-offs between reduced vulnerability and performance. Our overall goal is automatic vulnerability assessment and transformations that balance those concerns. To attain this goal, we are developing tools to inject faults into applications to identify vulnerable code regions and data structures, for which we develop techniques to reduce overall application vulnerability. Hand transformations demonstrate possible improvements and motivate directives to guide transformations that we automate through ROSE. That work naturally leads to an autotuning framework that explores combinations of vulnerability-reducing transformations and identifies the best overall set.
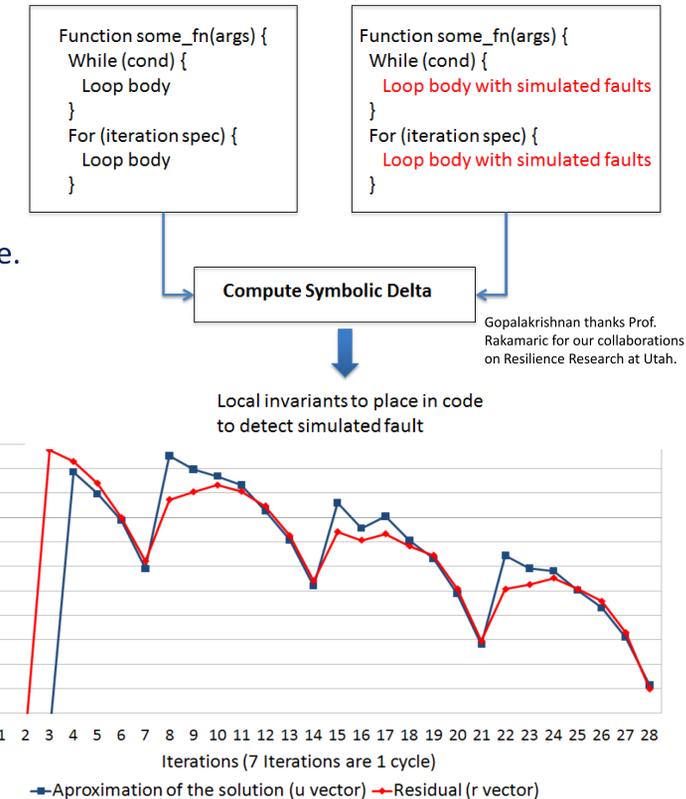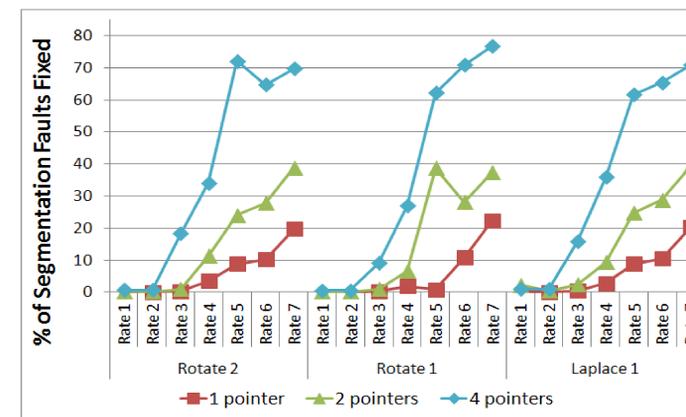
Vulnerability Assessment

- Fault injection campaigns.
- Identification of vulnerable code regions and data structures.

Setup Phase:
- Select coarse "grids,"
- Define interpolation, $P^{(m)}$, $m = 1, 2, \ldots$
- Define restriction and coarse-grid operators
  $R^{(m)} (= P^{(m)T})$ $A^{(m+1)} = R^{(m)} A^{(m)} P^{(m)}$

Solve Phase:

Relax $A^{(m)} u^m = f^m$

Compute $r^m = f^m - A^{(m)} u^m$

Restrict $r^{m+1} = R^{(m)} r^m$

Relax $A^{(m)} u^m = f^m$

Correct $u^m \leftarrow u^m + e^m$

Interpolate $e^m = P^{(m)} e^{m+1}$

Solve $A^{(m+1)} e^{m+1} = r^{m+1}$

Language Extensions & Compiler Technology for Resilience

- Annotations that allow user to express fault-tolerant requirements and expectations: when and where errors matter and what to do about them.
- Increase application MTBF, reducing the need for C/R (decrease bandwidth and energy)
- Resilience-aware Scheduling (RaS) leveraging idle resources to replicate instructions for transient error detection and correction, exploiting intrinsic resilience of application.
- ROSE source-level resilience-oriented and user-guided transformations

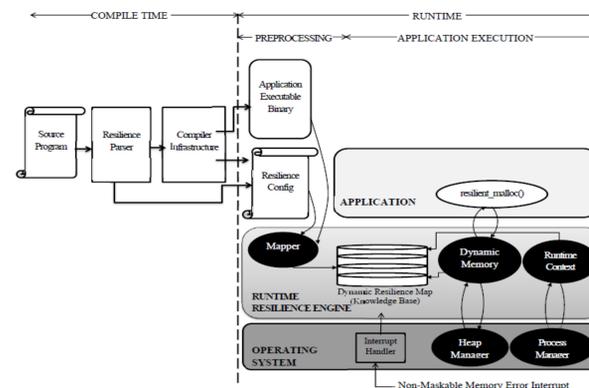## Resilience Improvements by Hand Transformations

- Replication of the most sensitive variables and routines.
- Error propagation studies.
- Code invariants derived through static analysis of the source code.
- Evaluation of the improvements through fault injection.


```
Function some_fn(args) {
    While (cond) {
        Loop body
    }
    For (iteration spec) {
        Loop body
    }
}

Function some_fn(args) {
    While (cond) {
        Loop body with simulated faults
    }
    For (iteration spec) {
        Loop body with simulated faults
    }
}
```

Compute Symbolic Delta

Gopalakrishnan thanks Prof. Rakamaric for our collaborations on Resilience Research at Utah.

Local invariants to place in code to detect simulated fault

General Autotuning Framework

- Vulnerability assessment of code regions and data-structures
- Code transformations through static and dynamic analysis
- Compiler techniques to automate and improve code transformations
- Optimization techniques to balance resilience and performance

Feedback: Optimal resilience improvements and their cost