The Global View Resilience Model

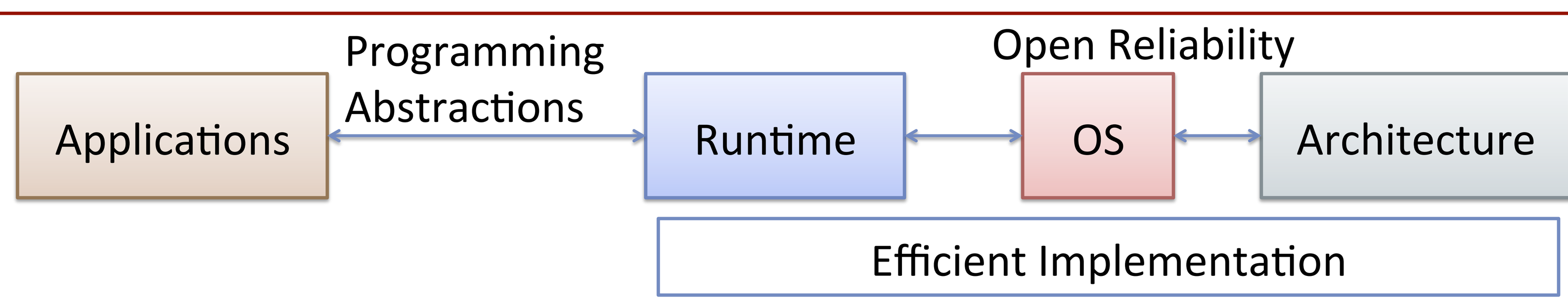
http://gvr.cs.uchicago.edu/

Zachary Rubenstein, Hajime Fujita, Guoming Lu, Aiman Fang, Ziming Zheng, Andrew A. Chien, *University of Chicago*;
Pavan Balaji, Kamil Iskra, Pete Beckman, James Dinan, Jeff Hammond, *Argonne National Laboratory*;
Robert Schreiber, *Hewlett--Packard Labs*

Background

- Widely accepted that Silicon scaling and low-voltage operation will produce rising error rates
- Need for a new programming model and a tool which address resilience issues

Goals



- Understand and create application-system partnership for flexible resilience
- Explore efficient implementation of resilient and multi-version data
- Create empirical understanding of GVR's effectiveness and performance requirements

Impact

- Resilient, globally-visible data store
- Incremental, portable approach to resilience for large-scale applications
- Flexible, application-managed cost and coverage for resilience

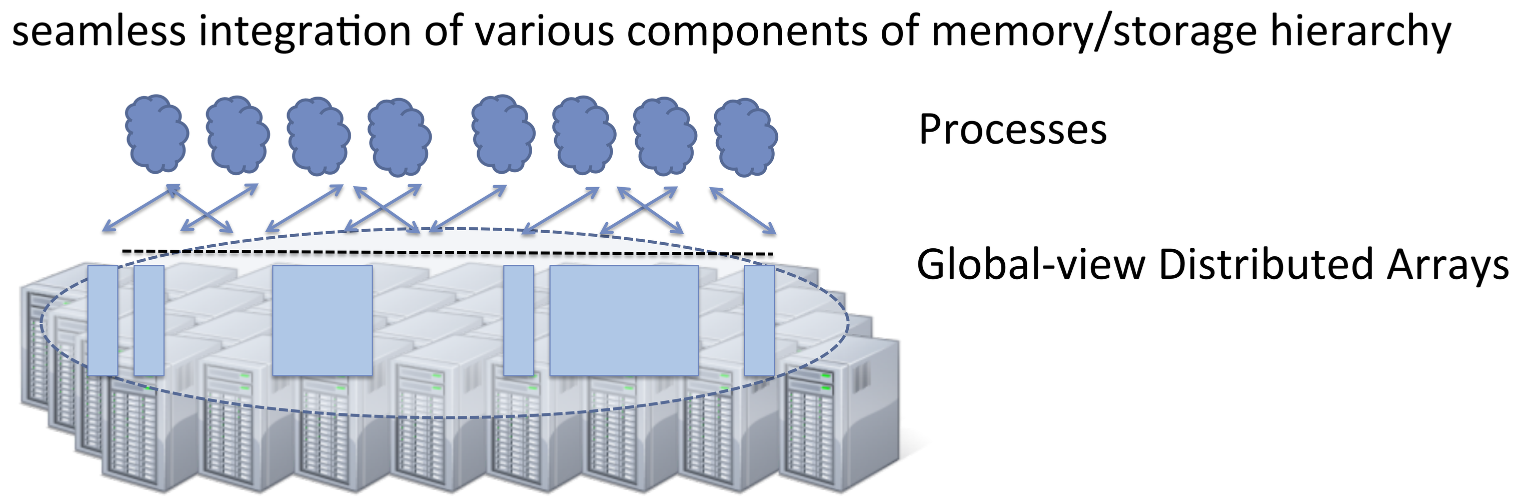
Research Challenges

- Understand application needs for flexible, portable resilience and performance
- Design of API suitable for use by application/library developers and tools
- Achieve efficient GVR runtime implementation for multi-version memory and flexible resilience
- Understand architecture support and its benefits
- Explore new opportunities created by GVR abstractions and its implementation technologies

Approach

GVR (Global View for Resilience)

- Exploits a global-view data model, which enables irregular, adaptive algorithms and exascale variability
- Provides an abstraction of data representation which offers resilience and seamless integration of various components of memory/storage hierarchy

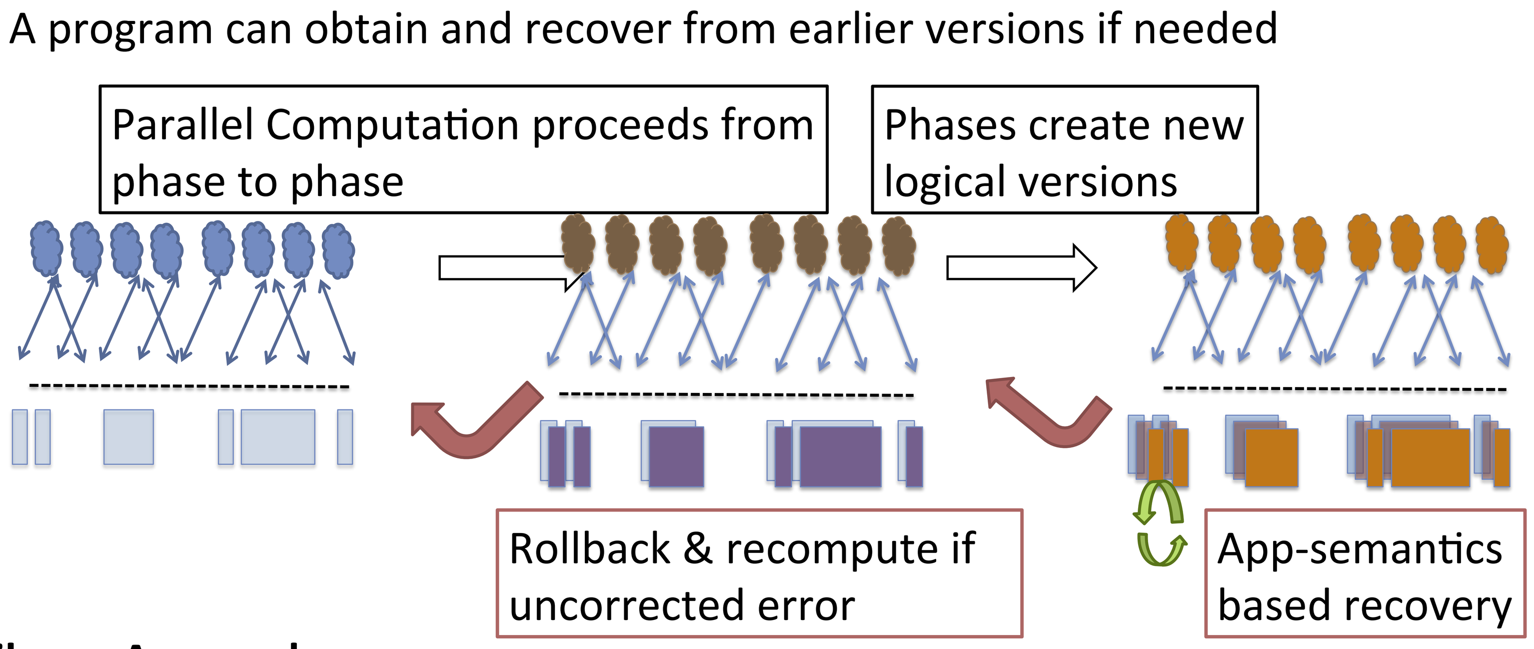


Non-uniform, Proportional Resilience

- Applications can specify which data are more important in order to manage reliability overheads

Multi-version Memory

- Computation phases form "versions" of data
- A program can obtain and recover from earlier versions if needed

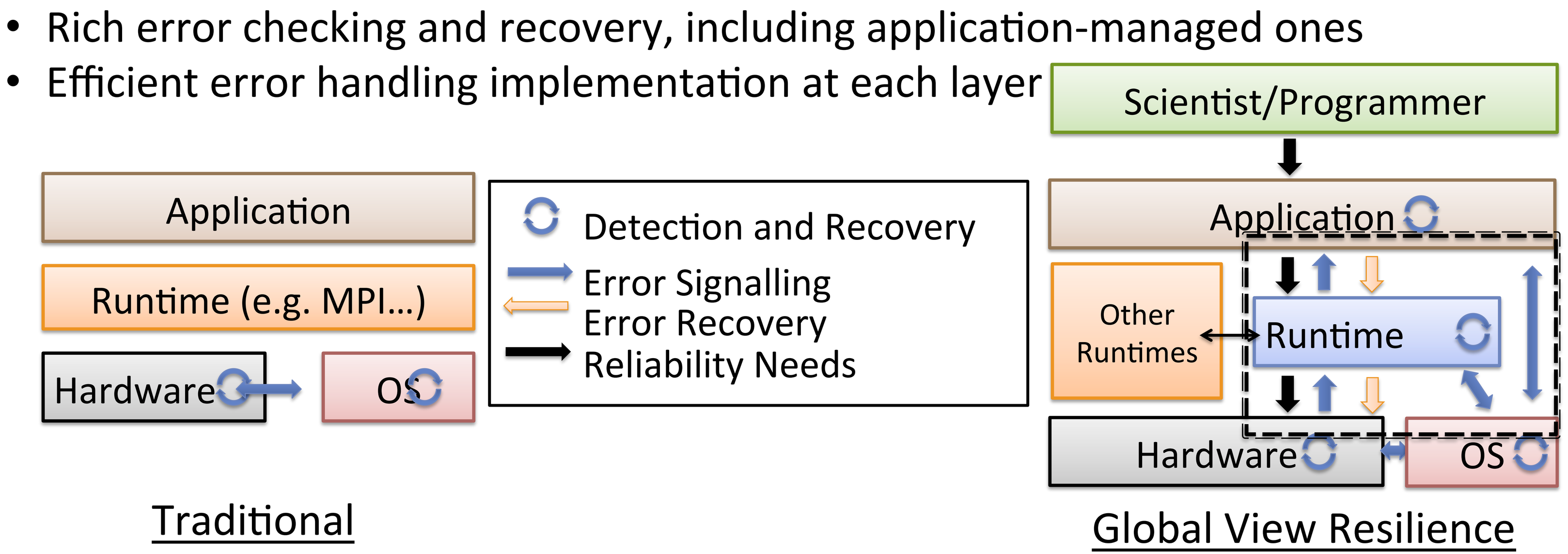


Library Approach

- Implemented as a library
- Can be used together with other libraries (e.g. MPI, Trilinos), allowing gradual migration to existing applications
- Can be a backend of other libraries/programming models (e.g. CnC, UPC, etc…)

Cross-layer Partnership (App, Runtime, OS, Architecture)

- Rich error checking and recovery, including application-managed ones
- Efficient error handling implementation at each layer



Example

- Pseudo-code from a molecular dynamics application

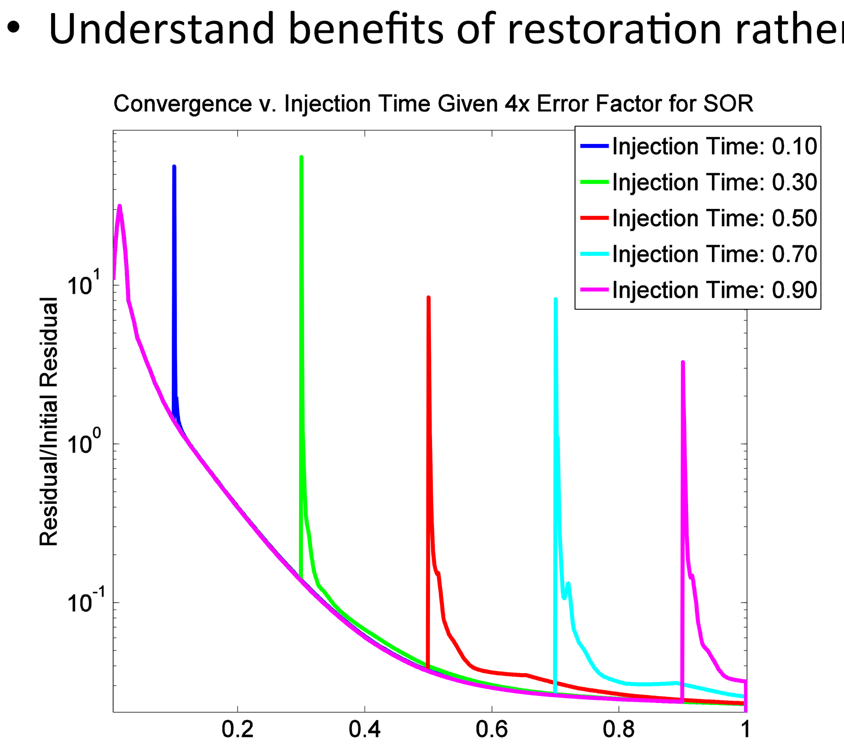
```
// Array creation and initialization
GDS_alloc(GDS_PRIORITY_HIGH, &gds);
GDS_register_global_error_handler(gds,
                                  errorhandler);
// full_check is a comprehensive and expensive
// check provided by a user
GDS_register_global_error_check(gds,
                                full_check);

void mainloop() {
  do_computation_and_communication(atoms);
  // lightweight error checking
  if (atoms_out_of_box(atoms)) {
    // switch to error handling mode
    GDS_raise_global_error(gds); }
  // preserve the important states
  GDS_put(atoms, gds);
  GDS_version_inc(gds); }

GDS_status_t errorhandler(GDS_gds_t gds) {
  // find a good version in the history
  do { GDS_move_to_prev(gds); }
  while (GDS_check_global_error(gds) != OK);
  // restore the states of atoms and resume
  GDS_get(atoms, gds);
  GDS_resume(gds); return OK;
}
```

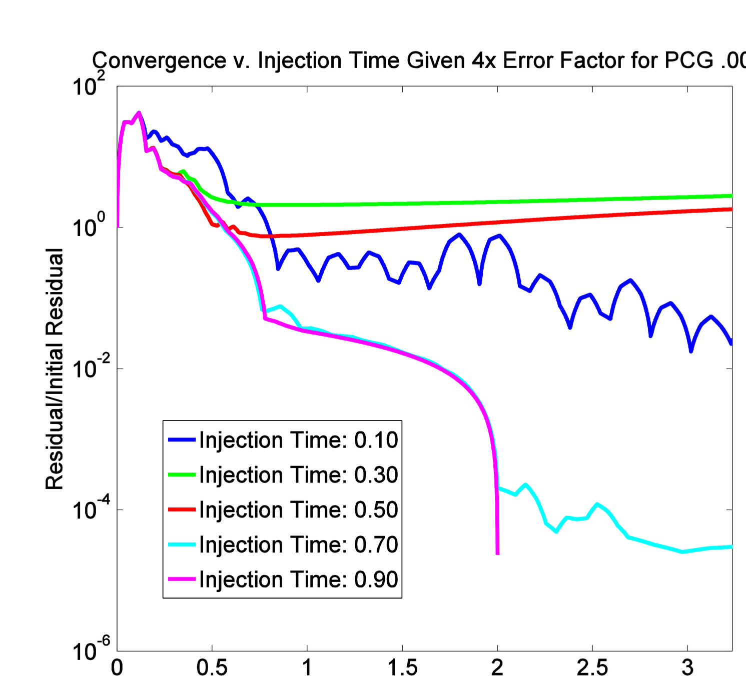
Progress and Accomplishments

- Use cases and initial design of GVR API
- Design of GVR runtime software architecture
- Initial research prototype of GVR, with multi-version array and application-managed error handling
- Functionality and performance explorations of user/kernel/hardware-based dirty bit tracking within the Local Reliable Data Store
- GVR-enabled two Mantevo mini-apps
- Modeling of multi-version checkpoint scheme that shows multi-version checkpoints critical for latent ("silent") errors
  - Please come and see our "**When is multi-version checkpointing needed?**" poster in Poster Session 2 for details.

Linear Solver Studies

- Inject errors of different severity at different points in computation for PCG and SOR
- Understand different methods for detecting injected errors
- Understand benefits of restoration rather than ignoring error





Future Efforts

- Fully-capable, robust implementation
- Efficient implementation of redundant, distributed global-view data structure
- Efficient multi-version snapshot (e.g. compression)
- Experiments with co-design applications
- Collaboration with OS/runtime community for cross-layer error handling

ASCR X-Stack Awards DE-SC0008603/57K68-00-145