Using Global View Resilience (GVR) to add Resilience to Exascale Applications

Hajime Fujita[1,2], Nan Dun[1,2], Aiman Fang[1], Zachary A. Rubenstein[1], Ziming Zheng[3], Kamil Iskra[2], Jeff Hammond[4], Anshu Dubey[5], Pavan Balaji[2], Andrew A. Chien[1,2]

[1]University of Chicago [2]Argonne National Laboratory [3]HP Vertica [4]Intel Labs [5]Lawrence Berkeley National Laboratory

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

Motivation

- Current high performance systems have achieved 10^{15} FLOPS and progress towards 10^{18} FLOPS.
- The exascale systems are comprised of millions of components, leading to higher error rates. It's anticipated that the mean time between failures (MTBF) could be less than an hour [1].
- Resilience becomes a major concern.
- Need for a new tool which address resilience issues.

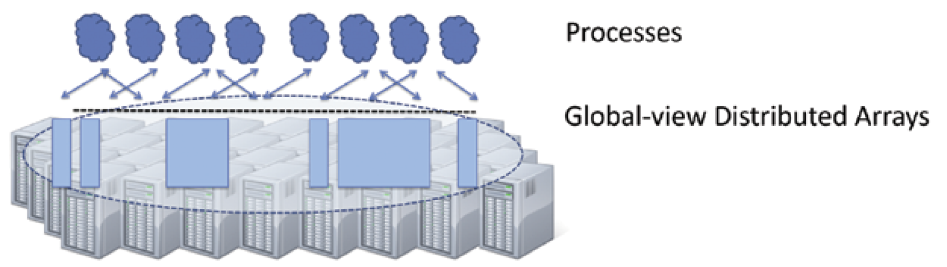
Global View Resilience

- A new library that exploits a global view data, and adds reliability to globally visible data [2, 3].
- Key features:
  - Multi-version, multi-stream distributed array: preserves critical application data with fine-grain manner, enables powerful recovery from complex errors such as latent errors
  - Open resilience: maximizes recoverable errors with cross-layer partnership, leverages application-level error handling with unified error handlers
- Portable, flexible, application-controlled resilience.
- Demonstrated usable, scalable resilience with gentle slope and flexible *forward error recovery*.
- Implemented as a library, which can be used together with other libraries (e.g. MPI, Trilinos), allowing gradual migration to existing applications, or as a backend of other libraries/programming models

Multi-version, Multi-stream, Distributed Arrays

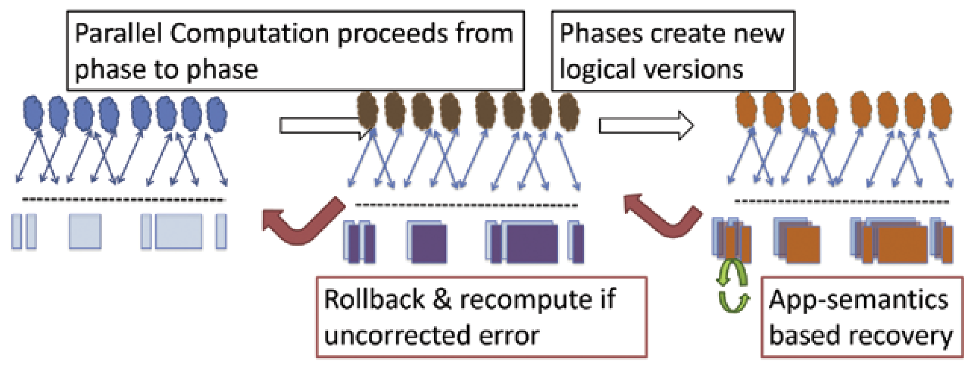
Global View

- 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



Multi-version, Multi-stream

- Computation phases form "versions" of data
- GVR array can preserve multiple versions upon application's request
- Application can retrieve arbitrary version for flexible recovery
- Having multiple versions is useful in many ways, e.g. rollback to old versions under presence of latent errors

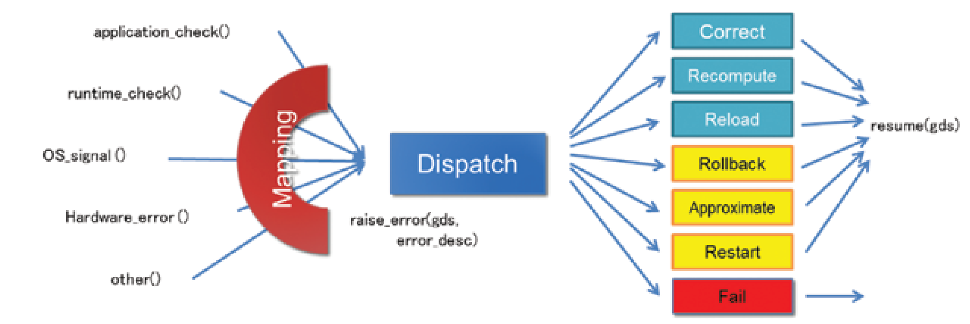


Non-uniform, Proportional Resilience

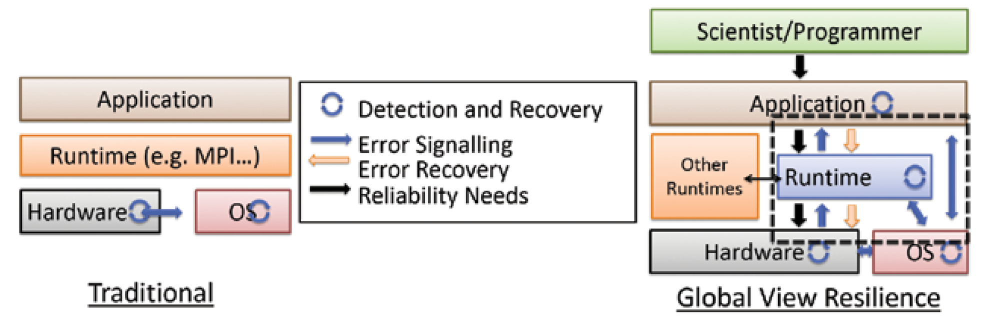
- Applications can specify which data are more important in order to manage reliability overheads
- Portable, controllable resilience
- Application-semantics based error detection and recovery

Open Resilience

- Unified Error Signaling and Handling
  - Various errors from different sources (e.g. HW, OS, runtime, application) are gathered into the GVR library, then dispatched to an application-defined error handler through the unified error signaling interface
  - Allows applications to supply their own error checking and handling handlers
  - Enables one error handler to be utilized for broader class of errors, leverages the effort spent for writing error handlers



- Cross-layer Partnership
  - Cross-layer collaboration among different components in the system maximizes the chance of error recovery, as application may be able to handle complex errors that cannot be handled in lower-level components (e.g. systems software or hardware)
  - Transition from traditional "fail-stop" model to "error signaling & handling" model



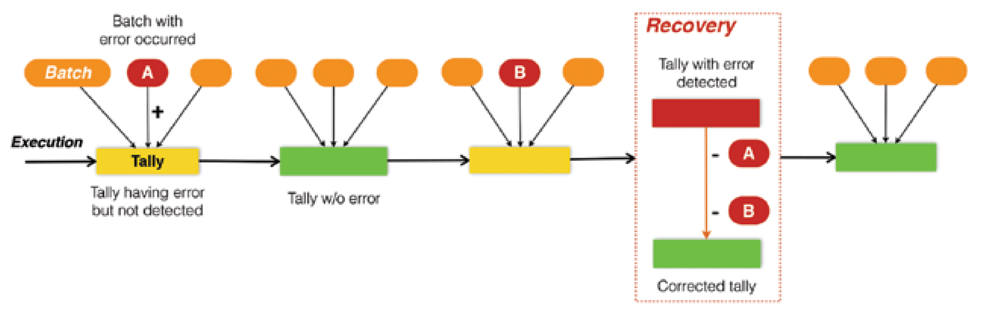
Rich Application Studies

- Molecular Dynamics: miniMD (SNL Mantevo Project), ddcMD (LLNL)
- Linear Solvers: miniFE (SNL Mantevo Project), PCG, GMRES
- Computation Library: Trilinos (SNL)
- Monte Carlo Neutron Transport: OpenMC (ANL CESAR co-design center)
- Adaptive Mesh Refinement Framework: Chombo (LBL)

GVR augmented ddcMD

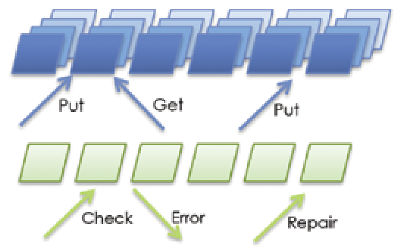
```
/* User defined error handler */
recovery_func(gds, error_descriptor) {
    GDS_get(local_data_structure, gds); /* Perform rollback */
    GDS_resume_global(gds, error_desc);
}

main() {
    /* Create global array data structure */
    GDS_alloc(&gds);

    GDS_create_error(&pred); ... /* Create error predicator */
    /* Register the specific error handler */
    GDS_register_global_error_handler(gds, pred, recovery_func);

    /* Molecular Dynamics Simulation Loop */
    simulation_loop() {
        /* Actual computation work */
        computation();
        if (error_detected()) { /* Error detection & signaling */
            /* Create error descriptor for the error */
            error_descriptor = GDS_create_error_descriptor(...); ...
            /* Raise the global error */
            GDS_raise_global_error(gds, error_descriptor);
            continue;
        }
        GDS_put(local_data_structure, gds);
        if (snapshot_point) { /* Take snapshot of correct states */
            GDS_version_inc(gds);
    }
}
```

Forward Error Recovery in OpenMC



APIs

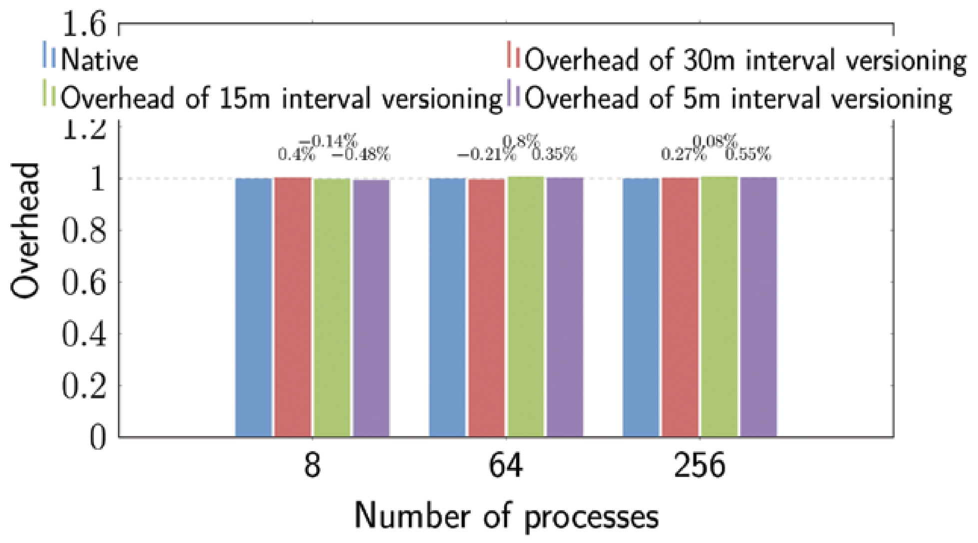
- Creating Global View structures
  - Create: GDS_alloc(), GDS_create()
- Global View Data Access
  - Data: GDS_put(), GDS_get()
  - Consistency: GDS_fence(), GDS_wait()
  - Accumulate: GDS_acc(), GDS_compare_and_swap()
- Versioning
  - Create: GDS_version_inc()
  - Navigate: GDS_get_version_number(), GDS_move_to_next(), GDS_move_to_prev(), GDS_move_to_newest().
- Error Signaling and Handling
  - Application checking, signaling, correction: GDS_register_global_error_handler(), GDS_register_local_handler()
  - System signaling, integrated recovery: GDS_raise_global_error(), GDS_raise_local_error()



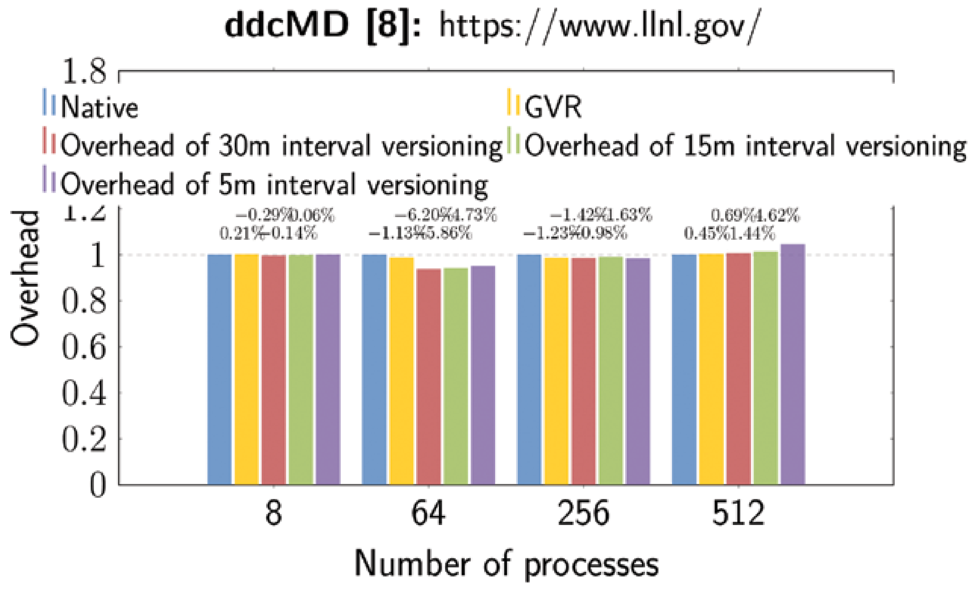
Performance Study

To measure the runtime overhead of GVR, experiments using OpenMC, ddcMD, and Chombo were conducted. Experiments for OpenMC and ddcMD were done on Midway, whereas the experiments for Chombo was conducted on NERSC Edison. As for the MPI library, MVAPICH2-2.0 on Midway and Cray MPT 7.0.0 on Edison were used.

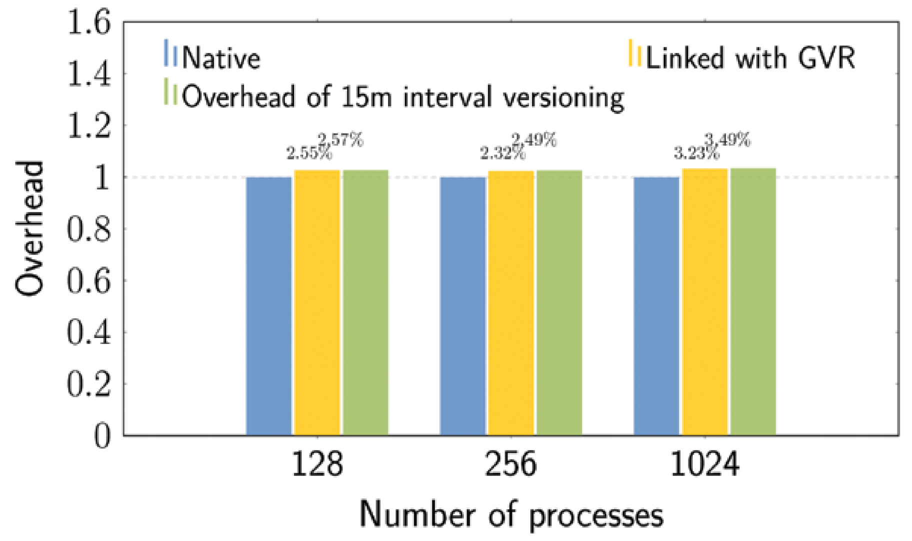
OpenMC [7]: http://mit-crpg.github.io/openmc/



ddcMD [8]: https://www.llnl.gov/



Chombo [9]: https://commons.lbl.gov/display/chombo/



Acknowledgments

We thank Mark Hoemmen, Mike Heroux, and Keita Teranishi for giving useful discussions on Trilinos and linear solvers, Brian van Straalen for advices on Chombo, Ignacio Laguna, David Richards for insights on ddcMD, and John R. Tramm, Andrew R. Siegel for supports on OpenMC.

GVR Gentle Slope

GVR can be easily applied to existing applications.

- No architectural changes
- Minimal (mostly <1%) code change

Code/App	Size (LOC)	Changed (LOC)	Change SW Architecture
Trilinos/PCG	300K	<1%	No
Trilinos/GMRES	300K	<1%	No
OpenMC	30K	<2%	No
ddcMD	110K	<0.3%	No
Chombo	500K	<1%	No

Summary

- GVR: Portable, flexible, application controlled resilience
  - Established model: use cases, extensive application partnership studies
  - Realized systems: several generations of prototypes, iteration informed by application studies
  - Gentle slope: <1% code change, negligible overhead
  - Scalable to exascale resilience: high error rates and latent and silent errors
- Application studies: Gentle slope, flexible forward error recovery
  - Numerous studies: incremental adoption, useful today
  - Compatible with existing software architectures
  - Enables exploitation of knowledge from all levels (app semantics-based recovery)
  - Enables all kinds of error recovery desired so far
- Maximize recoverable errors (open resilience)
  - Defined unified signaling and handling framework
  - Numerous examples of use
  - "Open resilience" can catalyze a cross-layer resilience eco-system

Future Work

- Efficient multi-version implementation, including efficient differences, compression, and efficient exploitation of NVRAM
- Work with community to establish Open Resilience APIs, infrastructure and portable error types/handling.
- Additional application studies, scalability
- Efficient portability studies, varying underlying hardware

References

[1] F. Cappello et al., *Toward exascale resilience.* International Journal of High Performance Computing Applications, 2009.

[2] The GVR Team, *Global View Resilience, API Documentation 0.8.1-rc0.* Technical Report. University of Chicago. April 28, 2014.

[3] The GVR Team, *How Applications use GVR: Use Cases.* Technical Report. University of Chicago. April 28, 2014.

[4] H. Fujita et al., *Log-Structured Global Array for Efficient Multi-Version Snapshots.* Submitted for publication, 2014.

[5] N Dun et al., *Data Decomposition in Monte Carlo Neutron Transport Simulations using Global View Arrays*, submitted for publication, 2014.

[6] Z. Zheng et al., *Fault Tolerance in an Inner-outer Solver: A GVR-enabled Case Study*, 11th International Meeting High Performance Computing for Computational Science-VECPAR, 2014.

[7] P. Romano and B. Forget, *The OpenMC Monte Carlo Particle Transport Code.* Annals of Nuclear Energy, 2013.

[8] F. Streitz et al., *Simulating Solidification in Metals at High Pressure: The Drive to Petascale Computing*, Journal of Physics: Conference Series, 2006.

[9] P. Colella et al., *Chombo Software Package for AMR Applications Design Document*, Lawrence Berkely National Laboratory, 2009.

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

We gratefully acknowledge the computing resources provided on Midway, high-performance computing cluster operated by the Research Computing Center at The University of Chicago. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.