GVR: Flexible, Portable, Scalable Recovery for Fail-stop and "Silent" Errors

Andrew A. Chien, The University of Chicago
and Argonne National Laboratory

Salishan Conference on HPC
April 28-30, 2015

Insights…

- Error rates for future systems are unknown, but could be as large as 30-70x higher from hardware alone
- Many errors in large-scale systems (not just HPC) come from software* (filesystems, runtimes, etc.)
- Application and library teams are creating innovative algorithmic approaches to detection and recovery with a broad view of error types and statistics.

- *May be growing due to increasing concurrent, asynchronous interactions (dynamic, adaptive)

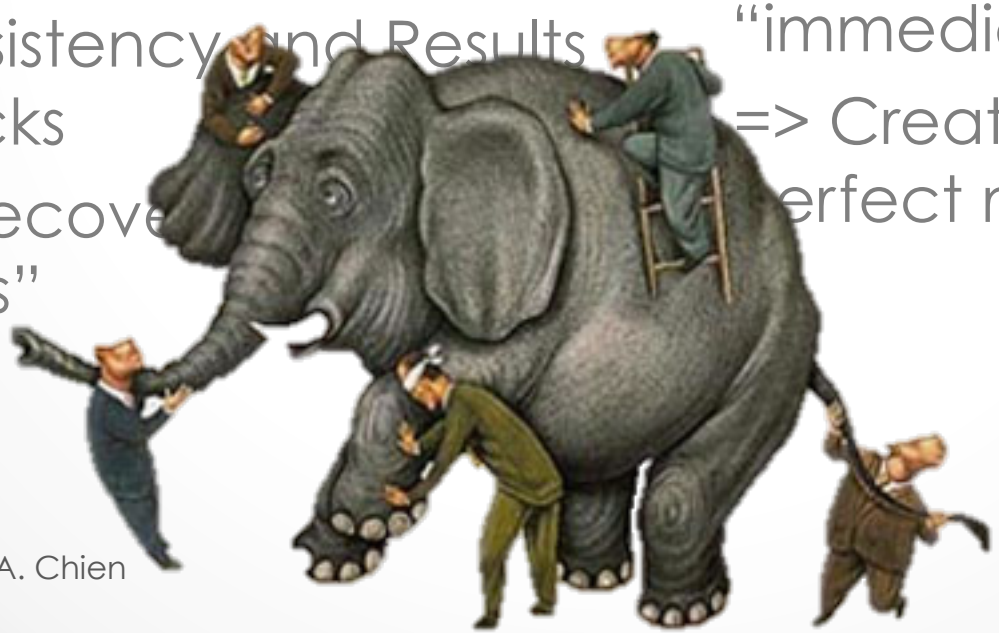
Two Views

Make Applications Resilient

- Algorithm-based Fault tolerance
- Application-based checkpointing
- Consistency and Results checks
- => Recove
errors"

Make Systems Resilient

- Checkpoint-Restart, Storage hierarchy, HW Systems Design
- => Recover "immediate" errors
- => Create Illusion of a erfect machine

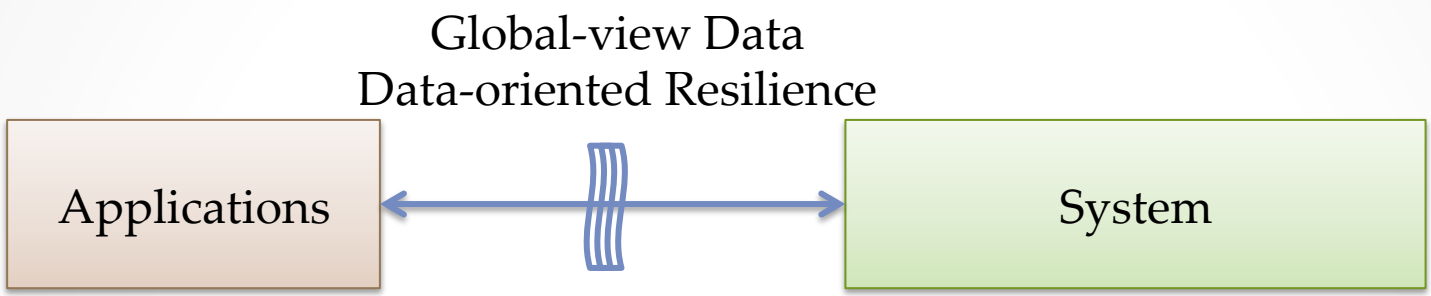


Just don't know exact shape of the resilience challenge! (Nathan's talk)

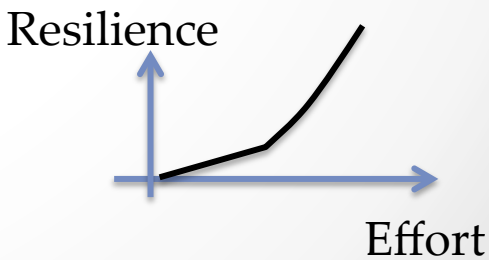
Outline

- GVR Approach and Flexible Recovery
- GVR in Applications Programming Effort
- GVR Versioning and Recovery Performance
- Summary
- ...More Opportunities with Versioning

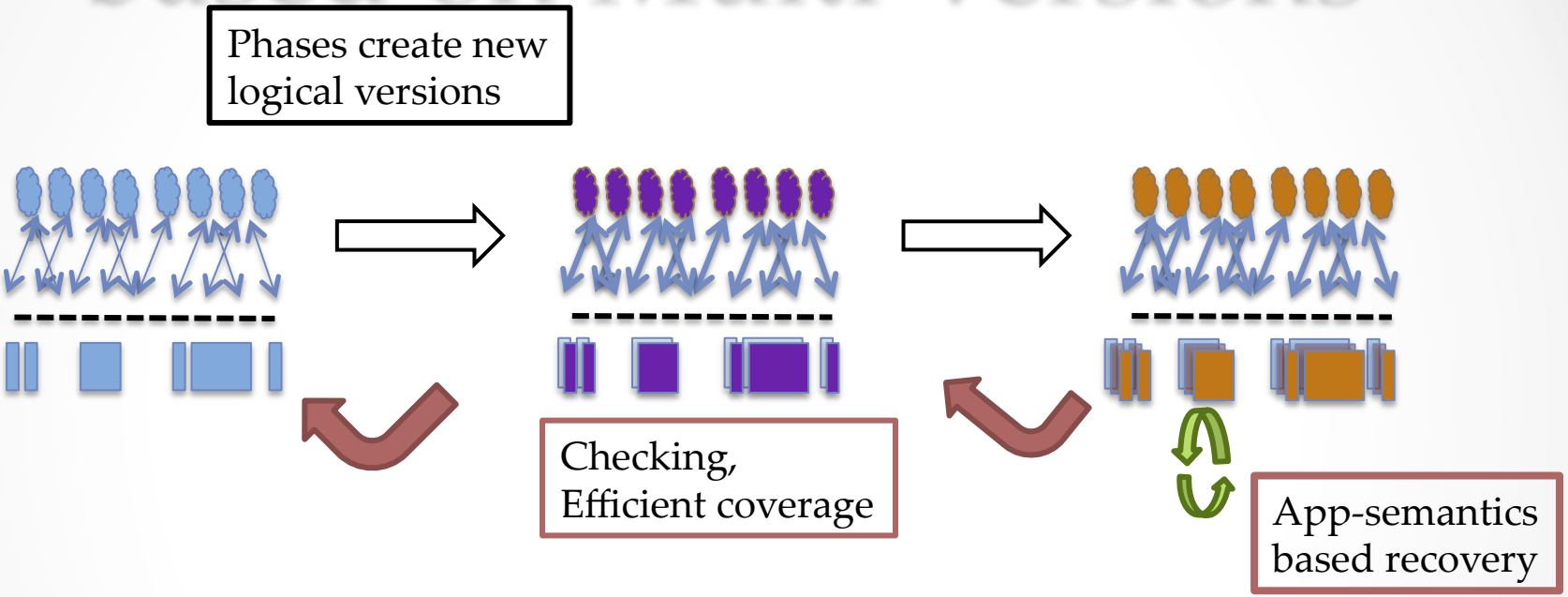
GVR Approach



- Application-System Partnership: System Architecture
  - Exploit algorithm and application domain knowledge
  - Enable "End to end" resilience model (outside-in), Levis' Talk
- Portable, Flexible Application control (performance)
  - Direct Application use or higher level models (task-parallel, PGAS, etc.)
  - GVR Manages storage hierarchy (memory, NVRAM, disk)
  - GVR ensures data storage reliability, covers error types
- Incremental "Resilience Engineering"
  - Gentle slope, Pay-more/Get-more, Anshu's talk



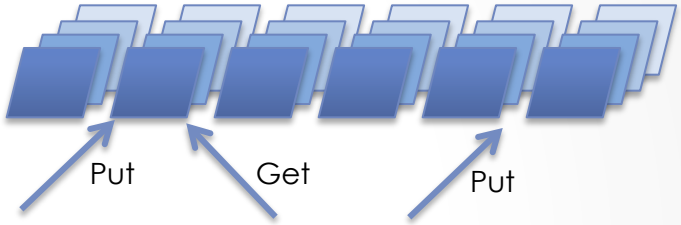
Data-oriented Resilience based on Multi-versions

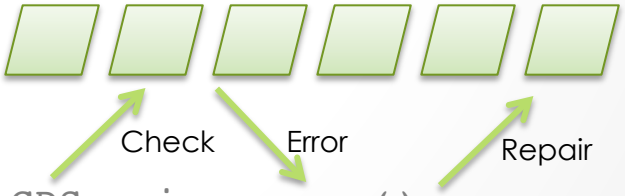


- Global-view data – flexible recovery from data, node, other errors
- Versioning/Redundancy customized as needed (per structure)
- Error checking & recovery framed in high-level semantics (portable)

GVR Concepts and API

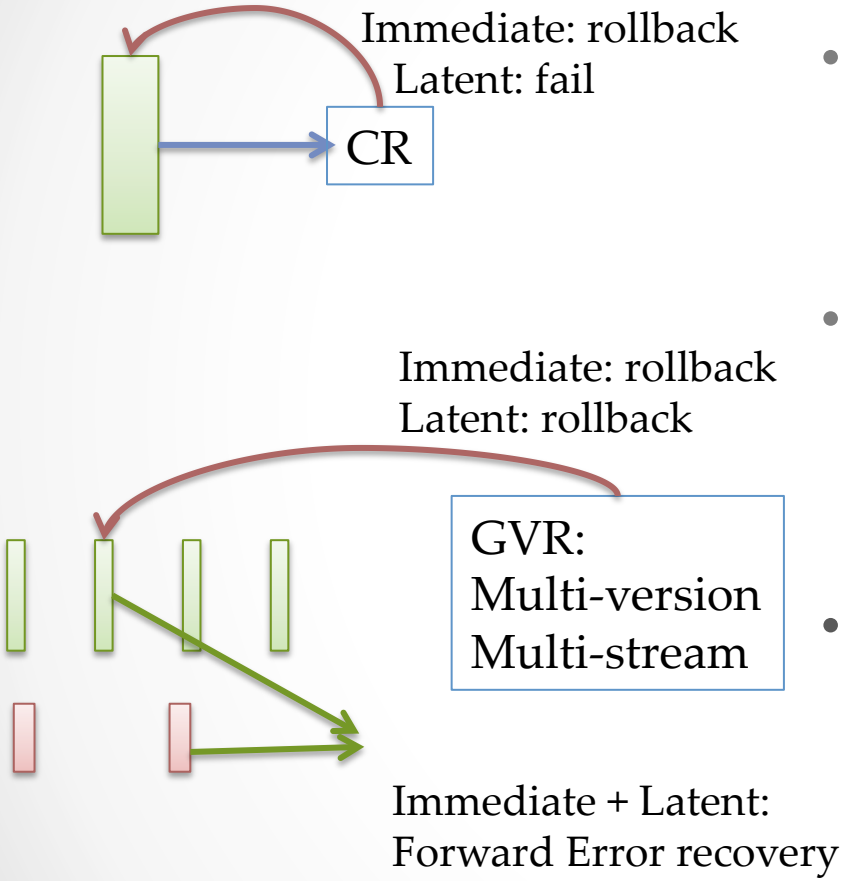
- Create Global view structures
  - New, federation interfaces
  - `GDS_alloc(...), GDS_create(...)`
- Global view Data access
  - Data: `GDS_put(), GDS_get()`
  - Consistency: `GDS_fence(), GDS_wait(),...`
  - Accumulate: `GDS_acc(), GDS_get_acc(), GDS_compare_and_swap()`
- Versioning
  - Create: `GDS_version_inc()`, Navigate: `GDS_get_version_number(), GDS_move_to_newest(), ...`
- Error handling
  - Application checking, signaling, correction: `GDS_raise_error(),`





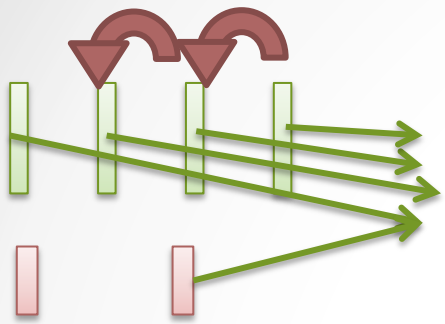
Applications have portable control over coverage and overhead of resilience.

GVR Flexible Recovery I

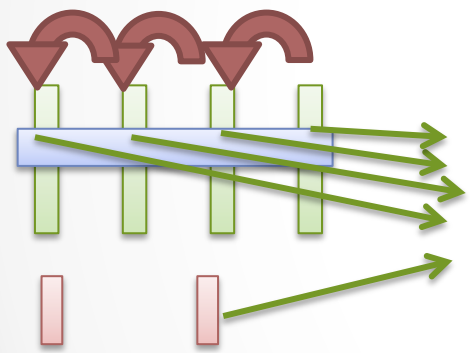


- Immediate errors: Rollback
- Latent/Silent errors: multi-version
  - Application recovery using multiple streams
- Immediate + Latent: novel forward error recovery
  - System or application recovery using approximation, compensation, recomputation, or other techniques
- Tune version frequency, data structure coverage, increased ABFT and forward error recovery for rising error rates

GVR Flexible Recovery II



Recovered Application



Recovered Application

- Complex errors, Rollback-diagnosis-forward
  - Flexible, Application-based recovery
  - Walk multiple versions
  - Diagnose
  - Compute corrections/approximations, execute forward
- Complex errors, Forward from multiple versions
  - Flexible, Application-based recovery
  - Partial materialization of multiple versions
  - Compute approximations, execute forward
- Tune version frequency, data structure coverage, increased ABFT and forward error recovery for rising error rates

GVR flexibility enables scalability across a wide range of error types and rates.

GVR Basic APIs

```
/* Add matrices C = A + B */
GDS_size_t counts[] = {N, N};
GDS_size_t lo[2], hi[2];
GDS_size_t ld[1] = N;
GDS_size_t min_chunk[2] = {1, N};
GDS_alloc(2 /*2-D*/, counts, min_chunk,
  GDS_DATA_DBL, GDS_PRIORITY_HIGH,
  GDS_COMM_WORLD, MPI_INFO_NULL, &gds_A);
/* Same for gds_B and gds_C */
/* Initialize A and B */
lo[0] = me; lo[1] = 0;
hi[0] = me; hi[1] = N-1;
GDS_get(my_A, ld, lo, hi, gds_A);
GDS_get(my_B, ld, lo, hi, gds_B);
GDS_wait(gds_A); GDS_wait(gds_B);
for (j = 0; j < N; j++)
  my_C[j] = my_A[j] + my_B[j];
GDS_put(my_C, ld, lo, hi, gds_C);
GDS_fence(gds_C);
```

Create 2-dimensional global arrays

Specifies a region to access from this process: process *i* accesses row *i*

Wait for non-blocking operations to complete

Global synchronization

GVR Versioning

```
/* Main computation loop */
do {
  sprintf(label, "version %d", i);
  do_computation(gds);
  GDS_version_inc(gds, 1,label, strlen(label))
} while (!converged);
```

User-defined label for version

Take current snapshot and increment version

```
/* Roll back from a correct version */
GDS_descriptor_clone(gds, &gds_clone);
do {
  GDS_move_to_prev(gds_clone);
} while (verify_contents(gds_clone) != OK);
GDS_get(buff, ld, lo, hi, gds_clone);
GDS_put(buff, ld, lo, hi, gds);
GDS_free(&gds_clone);
```

Make a cloned handle to the array for navigating versions

Search for a correct version

Copy old and correct data to the current version

Multiple versions enable more sophisticated recovery.

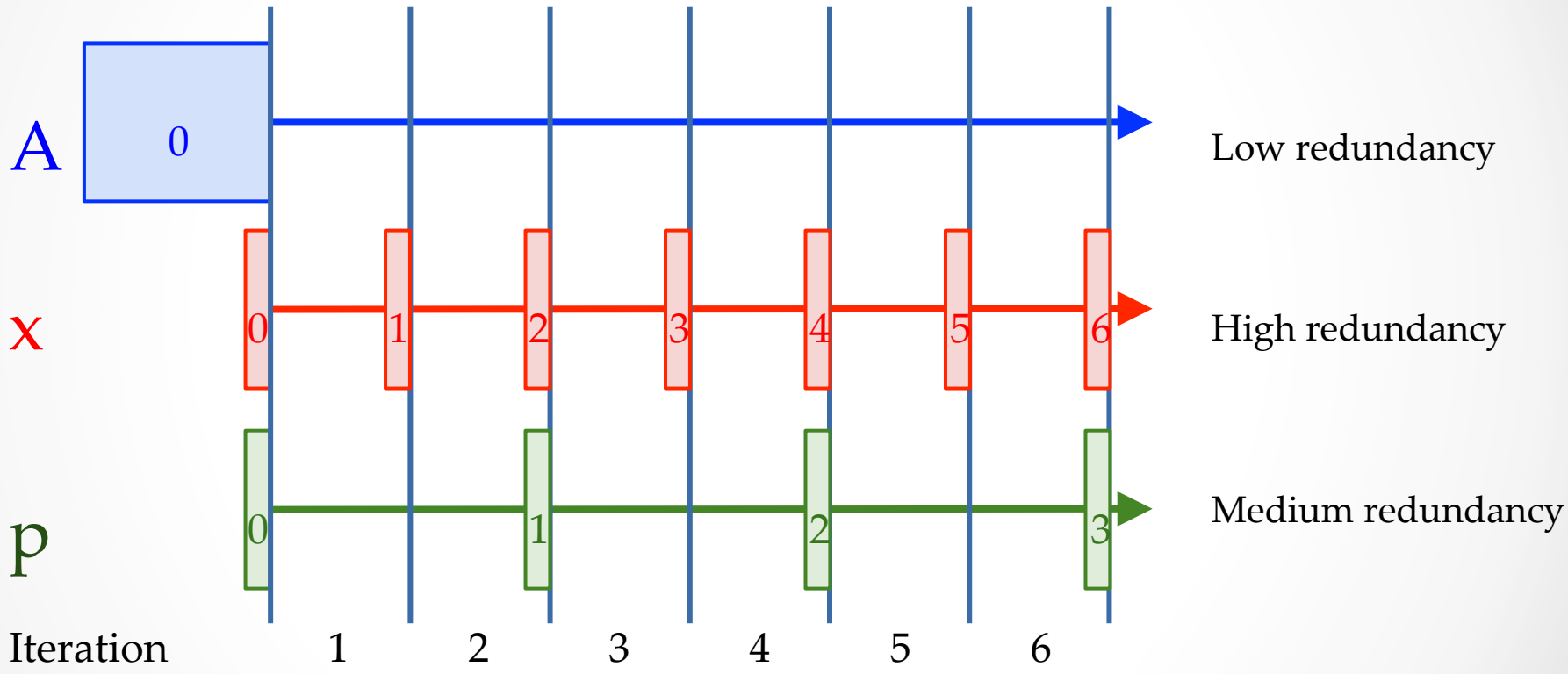
Simple Version Recovery: Preconditioned Conjugate Gradient

$A = \ldots$

$r = b - Ax$
iter $= 0$
while (iter $<$ max_iter) and $\|r\| >$ tolerance **do**
  iter $=$ iter $+1$
  $z = M^{-1}r$
  $\rho_{\text{old}} = \rho$
  $\rho = (r, z)$
  $\beta = \rho/\rho_{\text{old}}$
  $p = z + \beta p$
  $q = Ap$
  $\alpha = \rho/(p, q)$
  $x = x + \alpha p$
  $r = r - \alpha q$
end while

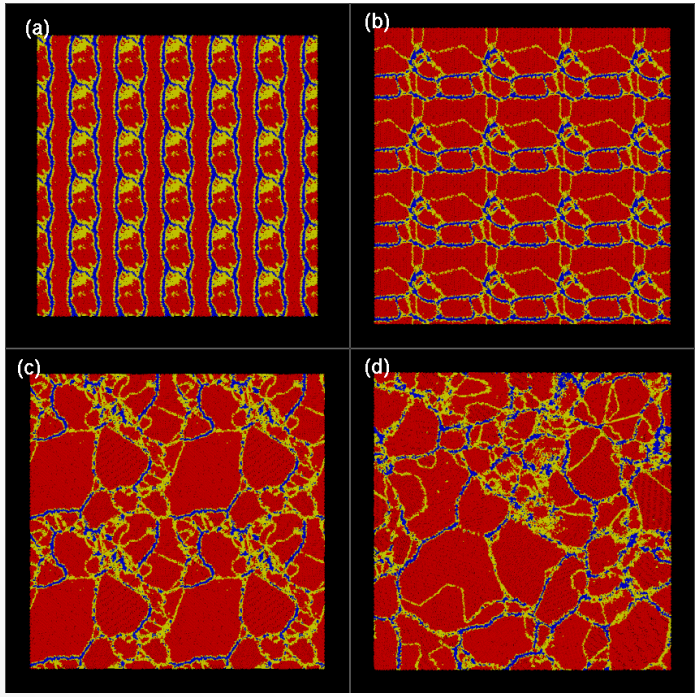
- Version x "solution vector"
  - Restore x on error
- Version p "direction vector"
  - Restore on error
- Version A "linear system"
  - Restore on error
- Restore from which version?
  - Most recent (immediately detected errors)
  - Older version (latent or "silent" errors)

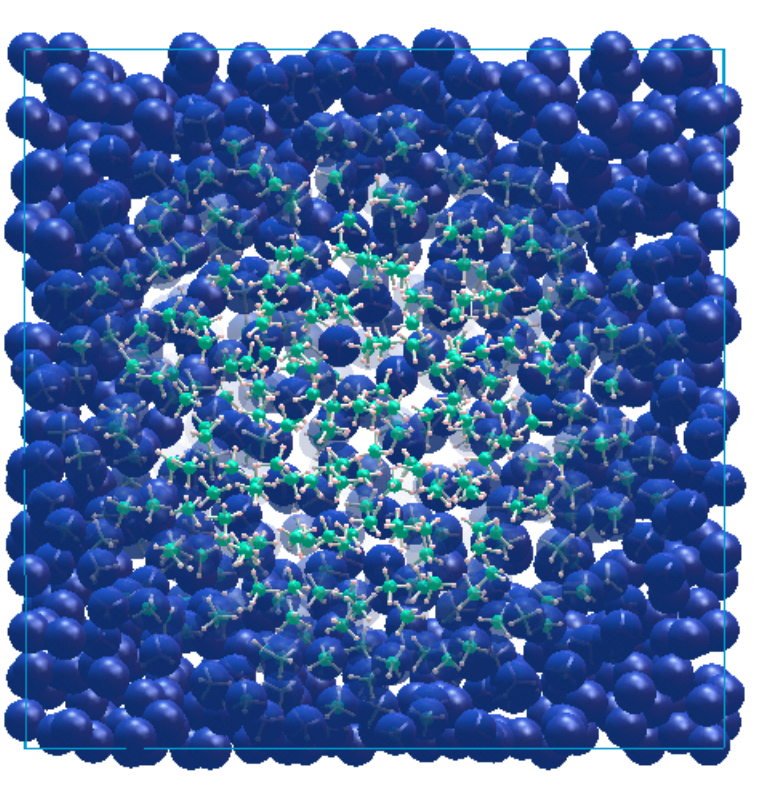
Multi-stream in PCG: Matching redundancy to need



Molecular Dynamics: miniMD, ddcMD

- miniMD: a SNL mini-app, a version of LAMMPS
- ddcMD is the atomistic simulation developed by LLNL -- scalable and efficient.





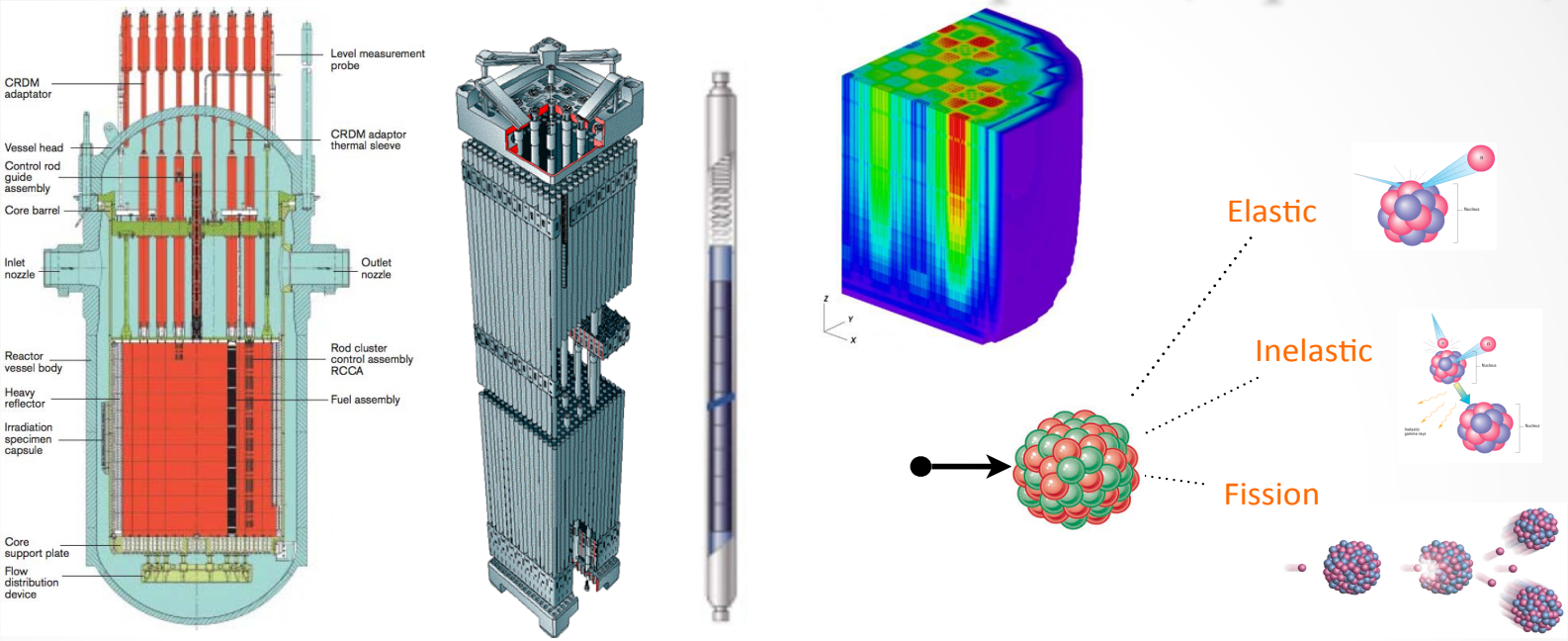
LLNL (Dave Richards & Ignacio Laguna)

ddcMD + GVR

```
main() {
    /* store essential data structures in gds */
    GDS_alloc(&gds);
    /* specify recovery function for gds */
    GDS_register_global_error_handler(gds, recovery_func);
    simulation_loop() {
        computation();
        error = check_func() /* finds the errors */
        if (error) {
            error_descriptor = GDS_create_error_descriptor(GDS_ERROR_MEMORY)
            /* signal error */
            /* trigger the global error handler for gds */
            GDS_raise_global_error(gds, error_descriptor);
        }
        if (snapshot_point){GDS_version_inc(gds);
                            GDS_put(local_data_structure, gds);};
    }
}
/* Simple recovery function, rollback */
recovery_func(gds, error_desc) {
    /* Read the latest snapshot into the core data structure */
    GDS_get(local_data_structure, gds);
    GDS_resume_global(gds, error_desc);
}
```


A. Fang, I. Laguna, D. Richards, and A. Chien. "Applying GVR to molecular dynamics: ..." CS TR-2014-04, Univ of Chicago.

Monte Carlo Neutron Transport (OpenMC)



- High fidelity, computation intensive and large memory (100GB~ cross sections and 1TB~ tally data)
- Particle-based parallelization is used with data decomposition
- Partition tally data by global array
- OpenMC: best scaling production code
- DOE CESAR co-design center "co-design application"

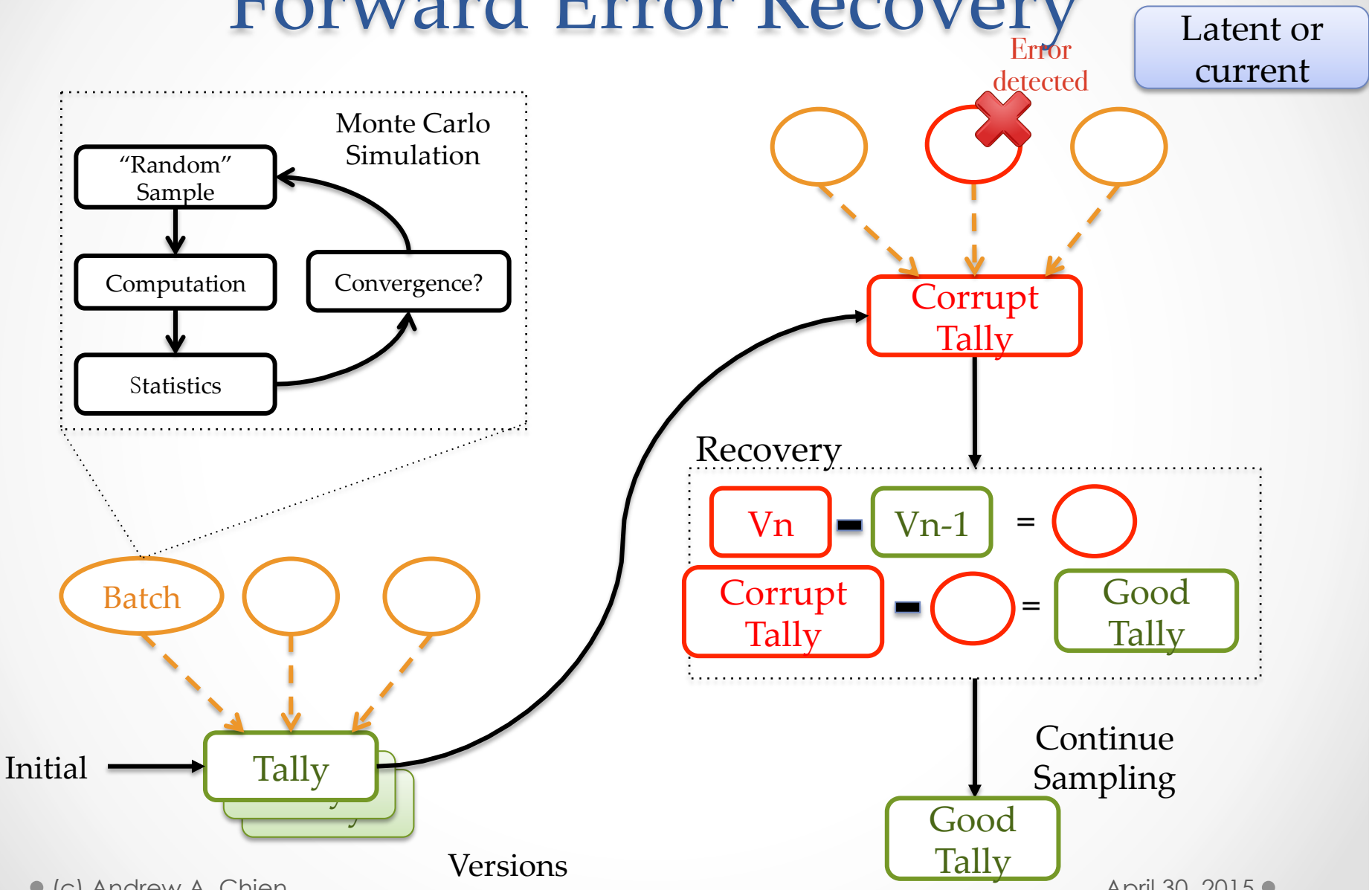
ANL/CESAR (Siegel, Tramm)

OpenMC + GVR

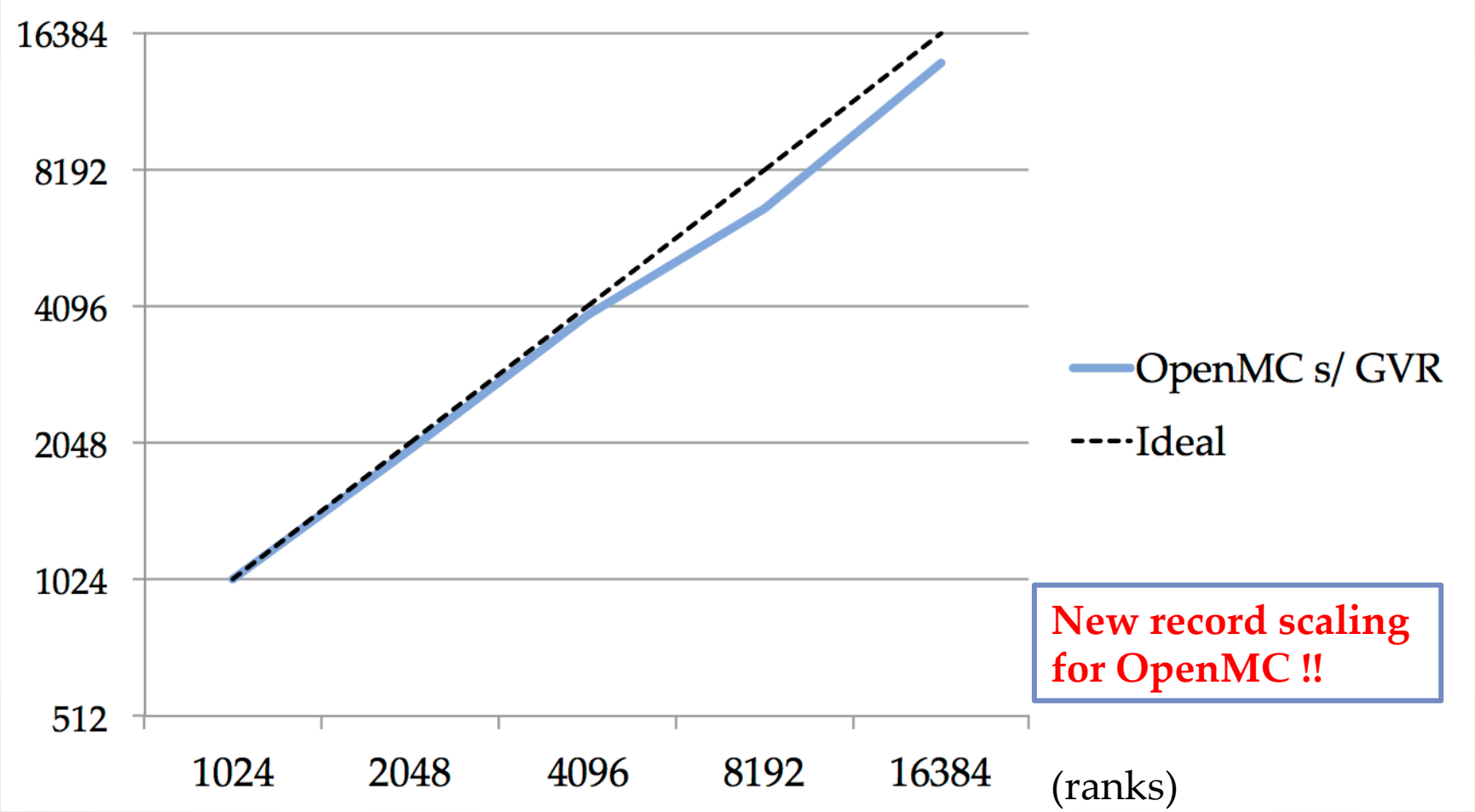
```
Initialize initial neutron positions
GDS_create(tally & source_site); //Create global tally array and source sites
for each batch
  for each particle in batch
     while (not absorbed)
         move particle and sample next interaction
         if fission
           GDS_acc(score, tally) // tally, add score asynchronously
           add new source sites
      end
         GDS_fence() // Synchronize outstanding operations
         resample source sites  & estimate eigenvalue
         if (take_version)  GDS_ver_inc(tally) // Increment version
                            GDS_ver_inc(source_site) // Increment version
   end
end
```

- Create Global view tallies
- Versioning: 259 LOC (<1%)
- Forward recovery: 250 (<1%)
- Overall application: 30 KLOC

Monte Carlo "Compensating" Forward Error Recovery



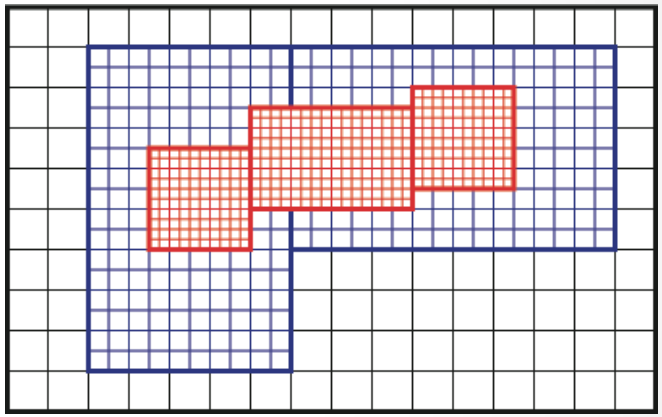
OpenMC+GVR Performance



N. Dun, H. Fujita, J. Tramm, A. Chien, and A. Siegel. Data Decomposition in Monte Carlo Neutron Transport Simulations using Global View Arrays, IJHPCA, May 2014

Chombo + GVR

- Resilience for core AMR hierarchy
  - Central to Chombo
  - Lessons applicable to Boxlib (ExaCT co-design app)
- Multiple levels, each with own time-step
- Data corruption and Process Crash Resilience
  - GVR used to version each level separately
  - Exploits application-level snapshot-restart
- GVR as vehicle to explore cost models for "resilience engineering" (Dubey)
  - Future: customize or localize recovery



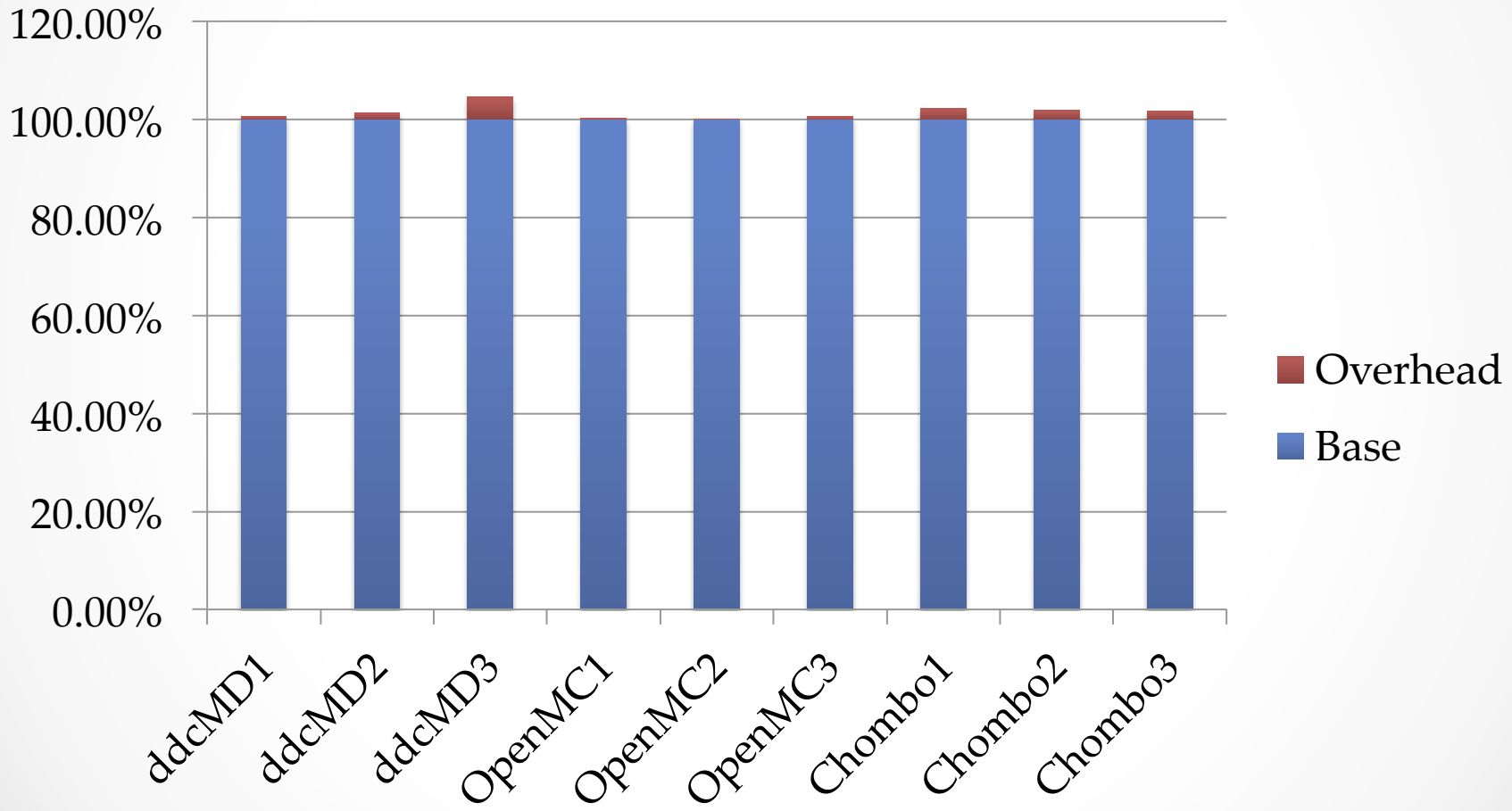
ExReDi/LBNL (Dubey, Van Straalen)

GVR Gentle Slope

Code/ Application	Size (LOC)	Changed (LOC)	Leverage Global View	Change SW architecture
Trilinos/PCG	300K	<1%	Yes	No
Trilinos/ Flexible GMRES	300K	<1%	Yes	No
OpenMC	30K	<2%	Yes	No
ddcMD	110K	<0.3%	Yes	No
Chombo	500K	<1%	Yes	No

GVR enables a gentle slope to Exascale resilience

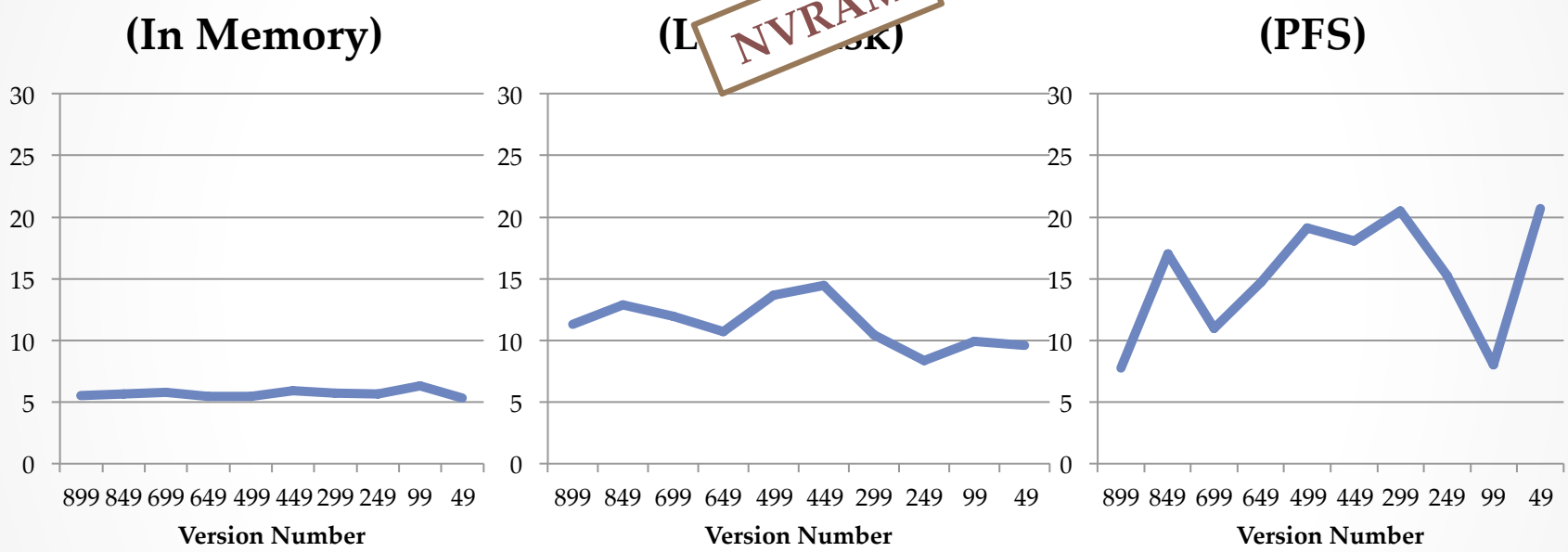
GVR Performance (Overhead)



Varied version frequency, against the native program. All < 2%.

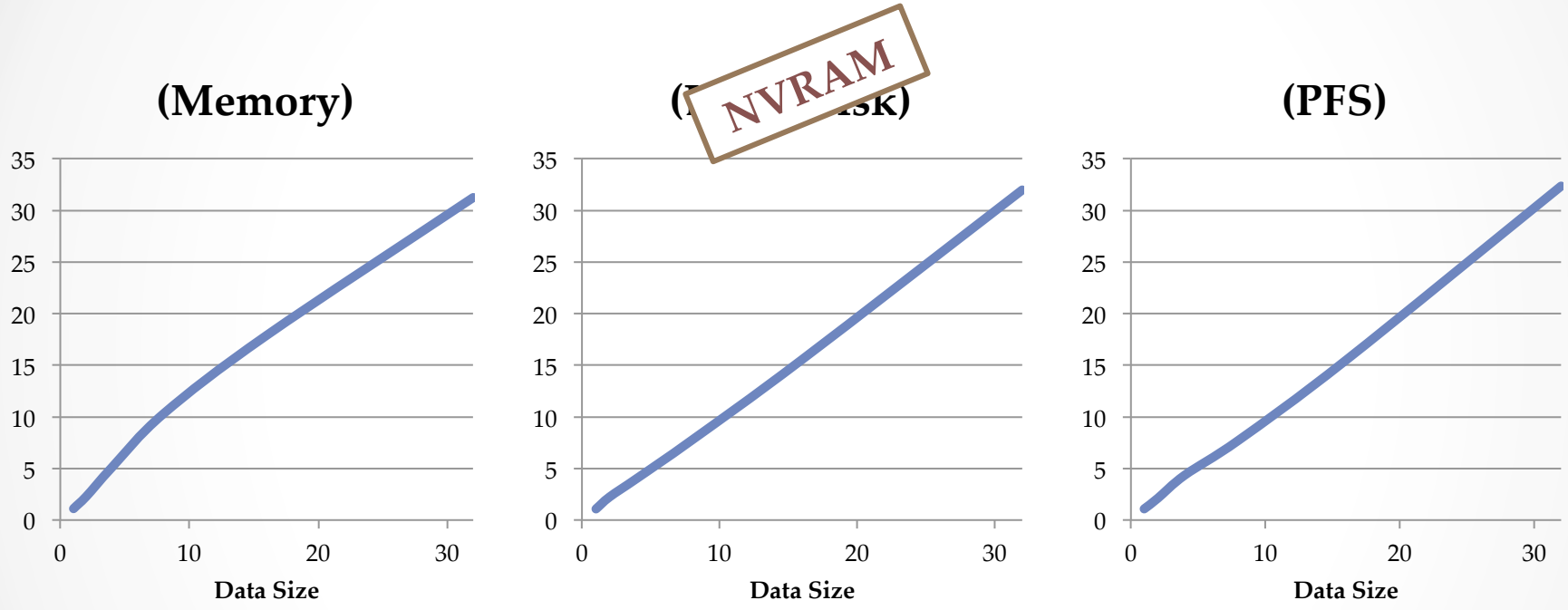
GVR Recovery Cost (Versions)

Novel Skew and MV representations



- Version management scales over # of version
  - Experiments: Versions 50 – 900
  - Subject to capacities and management of storage hierarchy

GVR Recovery Cost (Partial Reads)



- GVR does "partial materialization" efficiently
  - Cost proportional to data touched
  - Enables flexible multi-version recovery (temporal computation state)

GVR Summary

- Easy to add to an application
- Flexible control and coverage
- Flexible recovery (enables variety of forward techniques, approximations, etc.)
- Low overhead
- Efficient version restore (across versions)
- Efficient incremental restore

All Portable!

Additional GVR Research

…

Latent or "silent" error model

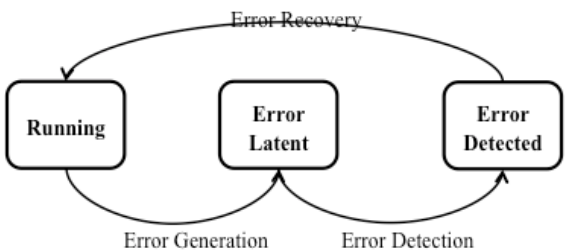
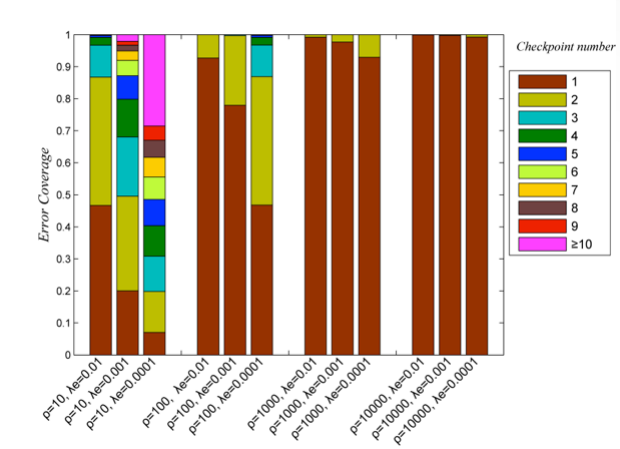
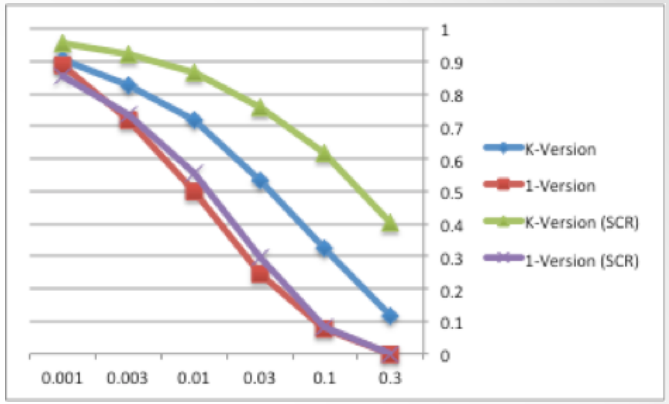


Fig. 1.b Latent Error Model



Multi-version critical for difficult to detect errors



Multi-version increases efficiency at high error rates

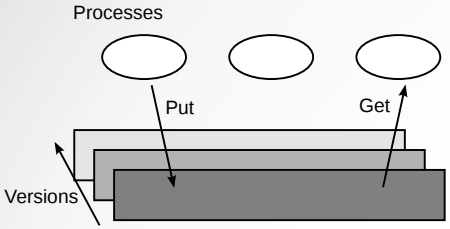
Latent Error Recovery

When multiple versions are useful

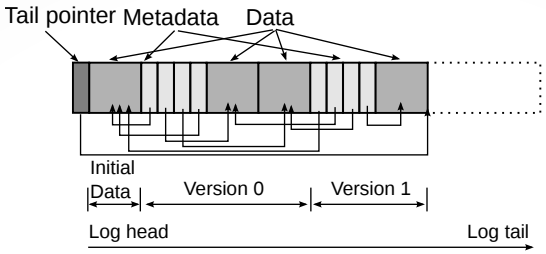
Impact on high-error rate regimes

Impact on difficult to detect errors

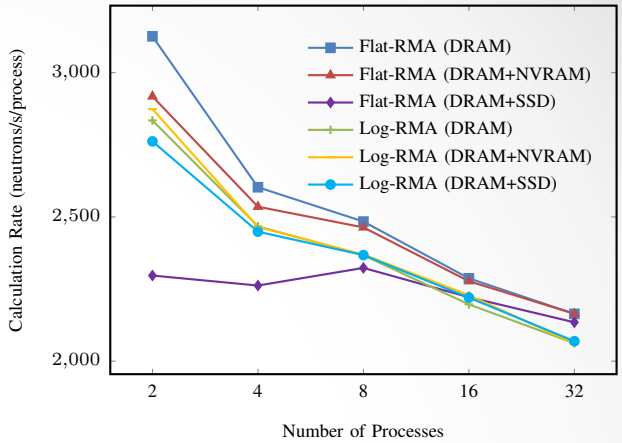
G. Lu, Z. Zheng, and A. Chien. When is multi-version checkpointing needed? 3rd Workshop on Fault-tolerance for HPC at extreme scale, FTXS '13, 2013.



Flat (Traditional)



Log-structured



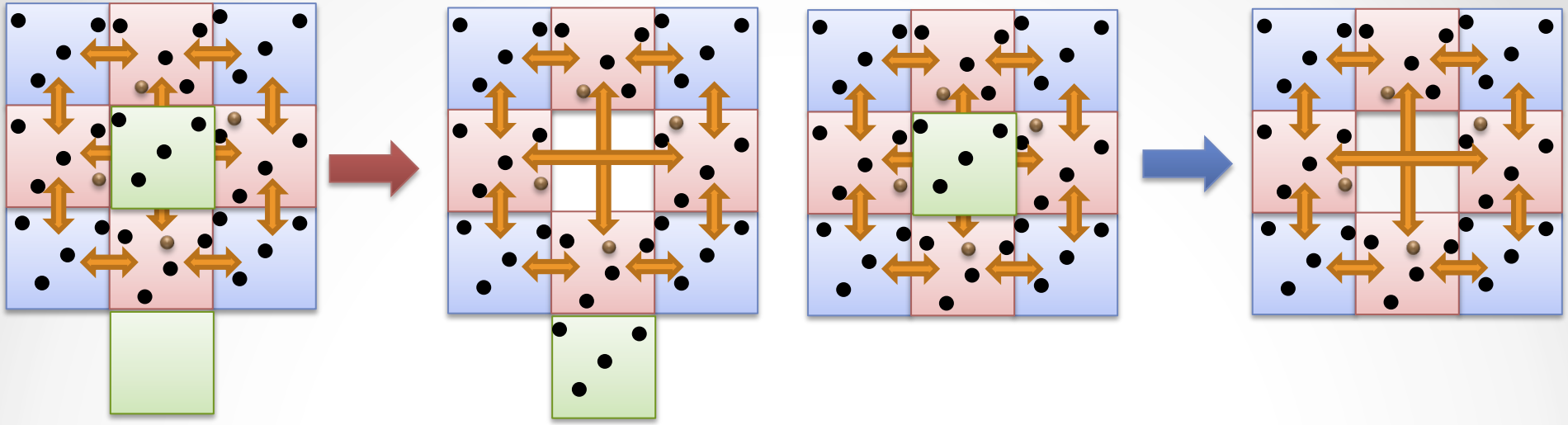
Comparative Studies with applications + varied memory hierarchies

Efficient Versioning

- Different implementations (SW, HW, OS, Application)
  - OS page tracking, dirty bits, SW declared
  - Skewed and Multi-version in-memory representations
- Efficient storage and materialization
- Leverages collective view
- Exploit NVRAM, burst buffers, etc.

H. Fujita, N. Dun, Z. Rubenstein, and A. Chien. Log-structured global array for efficient multi-version snapshots. CCGrid, May 2015.

H. Fujita, K. Iskra, P. Balaji, and A. Chien, "Empirical Characterization of Versioning Architectures", submitted.



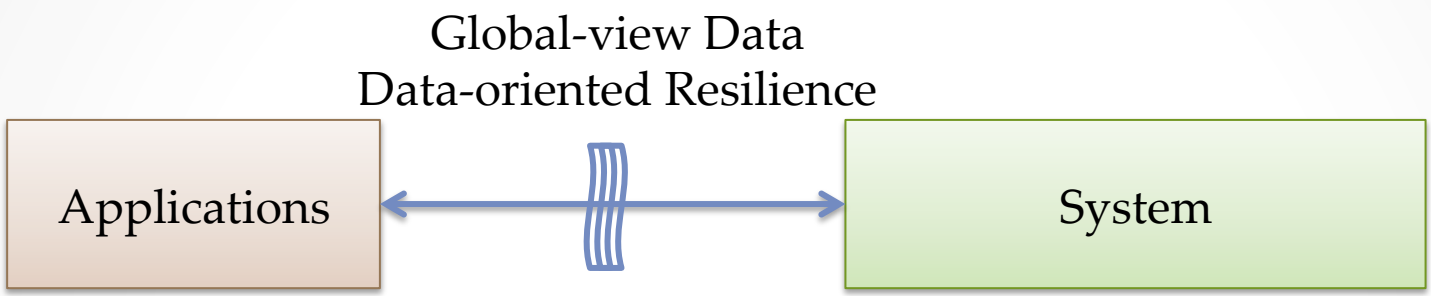
N+1->N and N->N-1 Recovery

- MPI Recovery (ULFM)
- Application Process Recovery
- Load Balancing and Performance
- Post-recovery Efficiency (PRE)

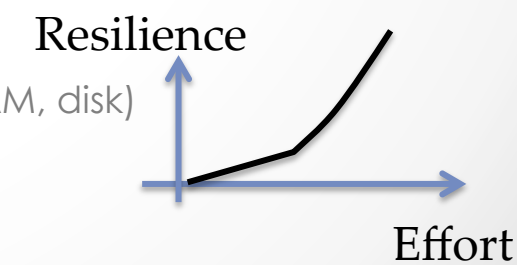
GVR Software Status

- Open source release, Oct 2014 (gvr.cs.uchicago.edu)
  - Tested with Miniapps – miniMD, miniFE experiments, and Full apps – ddcMD, PCG, OpenMC, Chombo
- Features
  - Versioned distributed arrays with global naming (a portable abstraction)
  - Independent array versioning (each at its own pace)
  - Reliable storage of the versioned arrays in memory, local disk/ssd, or global file system (thanks to Adam and SCR team!)
  - Whole version navigation and efficient restoration
  - Partial version efficient restoration (partial "materialization")
  - C native APIs and Fortran bindings
  - Runs on IBM Blue Gene, Cray XC, and Linux Clusters
- Key: all of the application investment is portable because the abstractions are portable

GVR's Version Namespace is a Scalable Abstraction for Resilience



- Application developers can exploit algorithm and domain knowledge
  - Apply "End to end" resilience model (outside-in) without disruptive code change
- GVR implementation provides efficient, portable resilience with control
  - GVR ensures data storage reliability, covers error types
  - Efficient management of storage hierarchy (memory, NVRAM, disk)
- Gentle slope "Resilience Engineering"



More GVR Info I

Basic API's and Usage

- GVR Team. Gvr documentation, release 0.8.1-rc0. Technical Report 2014-06, University of Chicago, Department of Computer Science, 2014.
- GVR Team. How applications use gvr: Use cases. Technical Report 2014-05, University of Chicago, Department of Computer Science, 2014.

GVR Architecture and Implementation Research

- Hajime Fujita, Kamil Iskra, Pavan Balaji, and Andrew A. Chien, "Empirical Characterization of Versioning Architectures", submitted for publication.
- Aiman Fang and Andrew A. Chien, "How Much SSD Is Useful for Resilience in Supercomputers", in IEEE Symposium on Fault-tolerance at Extreme-Scale (FTXS), June 2015.Hajime Fujita, Nan Dun, Zachary A. Rubenstein, and Andrew A. Chien. Log-structured global array for efficient multi-version snapshots. In CCGrid 2015..
- Guoming Lu, Ziming Zheng, and Andrew A. Chien. When is multi-version checkpointing needed? In Proceedings of the 3rd Workshop on Fault-tolerance for HPC at extreme scale, FTXS '13, pages 49–56, New York, NY, USA, 2013. ACM.
- Wesley Bland, Aurelien Bouteiller, Thomas Herault, Joshua Hursey, George Bosilca, and JackJ. Dongarra. An evaluation of User-Level Failure Mitigation support in MPI. Computing, 95(12):1171–1184, 2013.

More GVR Info II

Application Studies

- A. Chien, P. Balaji, N. Dun, A. Fang, H. Fujita, K. Iskra, Z. Rubenstein, Z. Zheng, J. Hammond, I. Laguna, D. Richards, A. Dubey, B. van Straalen, M Hoemmen, M. Heroux, K. Teranishi, A. Siegel. Exploring Versioning for Resilience in Scientific Applications: Global-view Resilience, submitted for publication, March 2015. (Best overall project summary)
- A. Chien, P. Balaji, P. Beckman, N. Dun, A. Fang, H. Fujita, K. Iskra, Z. Rubenstein, Z. Zheng, R. Schreiber, J. Hammond, J. Dinan, A. Laguna, D. Richards, A. Dubey, B. van Straalen, M Hoemmen, M. Heroux, K. Teranishi, A. Siegel, and J. Tramm, "Versioned Distributed Arrays for Resilience in Scientific Applications: Global View Resilience", in International Conference on Computational Science (ICCS 2015), Reykjavik, Iceland, June 2015.
- Nan Dun, Hajime Fujita, John R. Tramm, Andrew A. Chien, and Andrew R. Siegel. Data Decomposition in Monte Carlo Neutron Transport Simulations using Global View Arrays. Technical report, Department of Computer Science, University of Chicago, IJHPCA, April 2014.
- Aiman Fang and Andrew A. Chien. Applying gvr to molecular dynamics: Enabling resilience for scientific computations. Technical Report TR-2014-04, Department of Computer Science, University of Chicago, April 2014.
- Zachary Rubenstein, Hajime Fujita, Ziming Zheng, and Andrew Chien. Error checking and snapshot-based recovery in a preconditioned conjugate gradient solver. Technical Report TR- 2013-11, Department of Computer Science, University of Chicago, November 2013.
- Ziming Zheng, Andrew A. Chien, and Keita Teranishi. Fault tolerance in an inner-outer solver: A gvr-enabled case study. In 11th International Meeting High Performance Computing for Computational Science-VECPAR 2014, 2014.

Acknowledgements

- GVR Team: Hajime Fujita, Zachary Rubenstein, Aiman Fang, Nan Dun, Yan Liu (UChicago), Pavan Balaji, Pete Beckman, Kamil Iskra, (ANL), and application partners Andrew Siegel (Argonne/CESAR), Ziming Zheng (UC/Vertica), James Dinan (Intel), Guoming Lu (UESTC), Robert Schreiber (HP), Jeff Hammond (Argonne/ALCF/NWChem->Intel), Mike Heroux, Mark Hoemmen, Keita Teranishi (Sandia), Dave Richards (LLNL), Anshu Dubey, Brian Van Straalen (LBNL)

- SCR Team – some elements included in GVR system (thanks!)

- Department of Energy, Office of Science, Advanced Scientific Computing Research DE-SC0008603 and DE-AC02-06CH11357

- **For more information: http://gvr.cs.uchicago.edu/**