"Exploiting Global View for Resilience"

The Global View Resilience (GVR) Project

Andrew A. Chien, The University of Chicago and Argonne National Laboratory
Pavan Balaji, Argonne National Laboratory
X-stack PI Meeting @ MIT
May 28, 2014

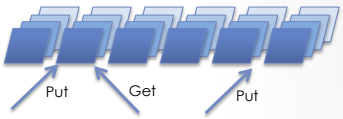
GVR Project Objectives (Vision)

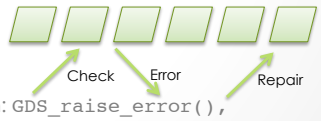
- Create and realize GVR model for Resilience: Portable, Flexible, Application-Controlled Resilience
- Application Studies: Demonstrate usable, scalable resilience with Gentle Slope and Flexible forward error recovery
- Maximize recoverable errors (x-layer resilience)

Create a gentle-slope to Exascale resilience

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(), GDS_register_local_error_handler()...`
  - System signaling, integrated recovery: `GDS_raise_error(), GDS_resume()`





GVR Vision + Progress

- GVR Model: 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 (5 demonstrations, <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 (app, library, programming system)
  - Enables exploitation of knowledge from all levels (app semantics-based recovery)
  - Enables all kinds of error recovery desired so far
- Maximize Recoverable Errors (cross-Layer)
  - Defined Unified signaling and Handling framework
  - Numerous Examples of use
  - "Open Resilience" can catalyze a cross-layer resilience eco-system

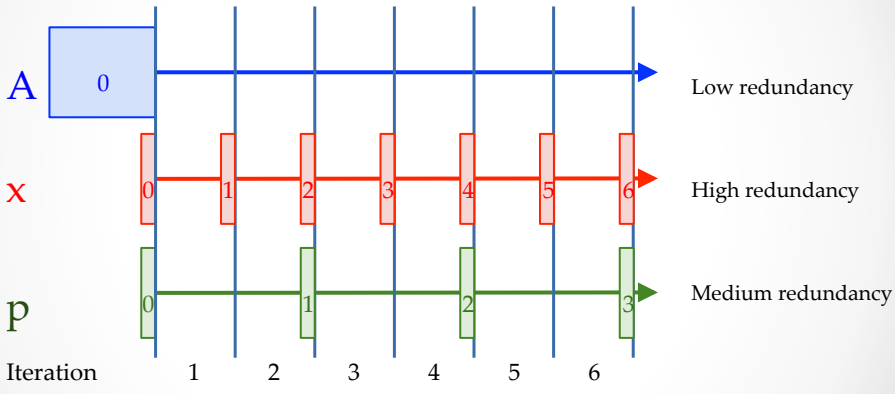
Simple Version Recovery: Preconditioned Conjugate Gradient

$A = \ldots$

1: $r = b - Ax$
2: iter $= 0$
3: **while** (iter $<$ max_iter) and $\|r\| >$ tolerance **do**
4: iter $=$ iter $+1$
5: $z = M^{-1}r$
6: $\rho_{\text{old}} = \rho$
7: $\rho = (r, z)$
8: $\beta = \rho/\rho_{\text{old}}$
9: $p = z + \beta p$
10: $q = Ap$
11: $\alpha = \rho/(p, q)$
12: $x = x + \alpha p$
13: $r = r - \alpha q$
14: **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



Applying GVR to Flexible GMRES

Joint w/ Mark Hoemmen, Keita Teranishi, and Mike Heroux of SNL

Input: Linear system $Ax = b$ and initial guess x_0.
Output: Approximate solution x_m.

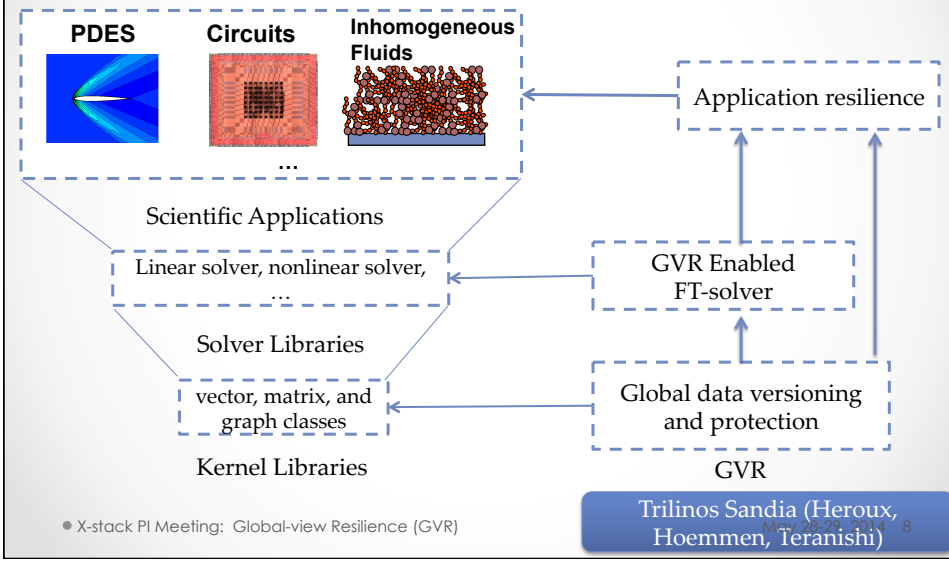
1: $r_0 := Ax - b, \beta := ||r_0||_2, q_1 := r_0/\beta$
2: **for** $j = 1, \ldots, m$ **do**
3: Inner solver for inexact solution z_j in $q_j = Az_j$ → Robust to minor Inner Solver errors
4: $v_{j+1} := Az_j$
5: **for** $i = 1, \ldots, j$ **do**
6: $H(i,j) := (v_{j+1}, q_i)$
7: $v_{j+1} := v_{j+1} - q_i H(i,j)$
8: **end for**
9: $H(j+1,j) := ||v_{j+1}||_2$
10: $q_{j+1} := v_{j+1}/H(j+1,j)$
11: $y_j := argmin_y ||H(1:j+1, 1:j)y - \beta e_1|||_2$
12: $x_j := x_0 + [z_1, \ldots, z_j]y_j$
13: **end for**
14: **if** converged **then**
15: Return x_m
16: **else**
17: $x_0 := x_m$, *go to* 1
18: **end if**

Check: Residual decreasing

Remove error in outer solver
- Restart, recompute
- Version recovery
- Various error rates

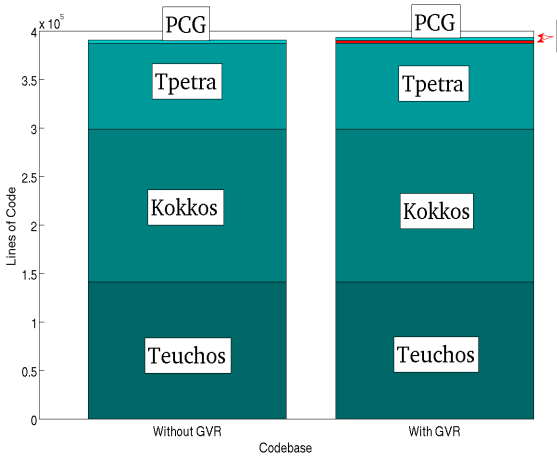
Ziming Zheng, Andrew A. Chien, Keita Teranishi, "Fault Tolerance in an Inner-Outer Solver: a GVR-enabled Case Study", in Proceedings of VECPAR 2014, July 2014.

Trilinos Library Hierarchy + GVR



Trilinos Sandia (Heroux, Hoemmen, Teranishi)

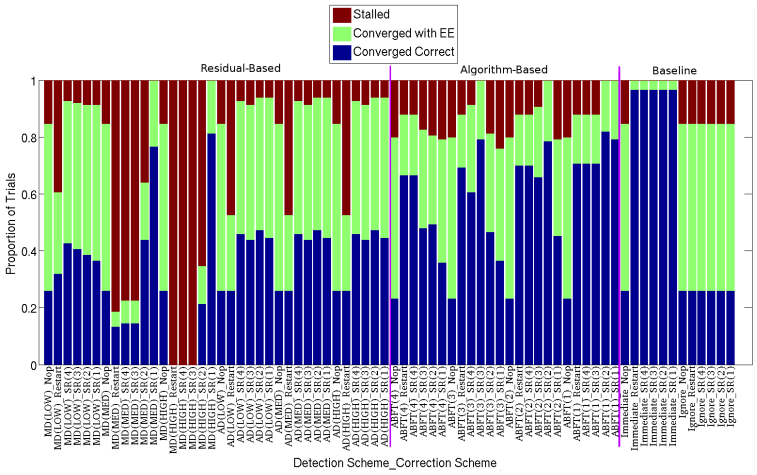
GVR+Trilinos: Gentle Slope Resilience



- <1% Trilinos library Code change (C++)
- 23 lines PCG Solver change
- 25 lines Flexible GMRES change

GVR's design enables application and library resilience with small code change.

Overall Outcomes: Stalls, Convergence, and EE



- Immediate mostly converges correct
- Nop admits a high probability of EE
- Residual-based trades low EE for high stall
- High-frequency ABFT has low EE and low stall

PCG Study, Rubenstein MS Thesis, March 2014

Latent or "silent" error model

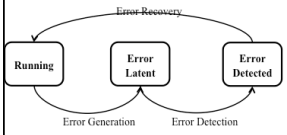
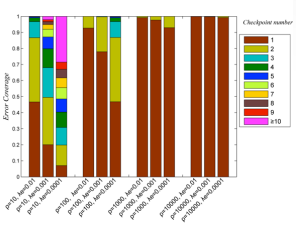
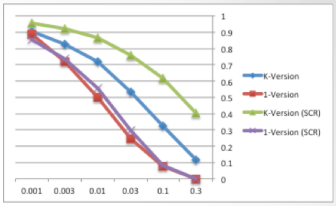


Fig. 1.b Latent Error Model



Multi-version critical for difficult to detect errors

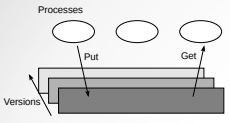


Multi-version increases efficiency at high error rates

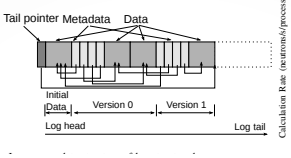
Latent Error Recovery

When multiple versions are useful
Impact on high-error rate regimes
Impact on difficult to detect errors

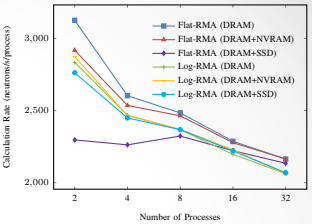
G. Lu, Z. Zheng, and A. Chien. When is multi-version checkpointing needed? 3rd Workshop on Fault-tolerance for HPC at extreme scale, FTXS '13, 2013.



Flat (Traditional)



Log-structured



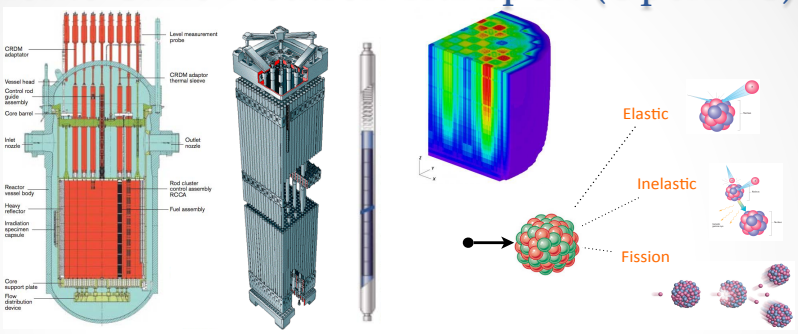
Comparative Studies with applications + varied memory hierarchies

Efficient Versioning

- Different implementations (SW, HW, OS, Application)
- 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. Uchicago CS Tech Report, May 2014

Monte Carlo Neutron Transport (OpenMC)



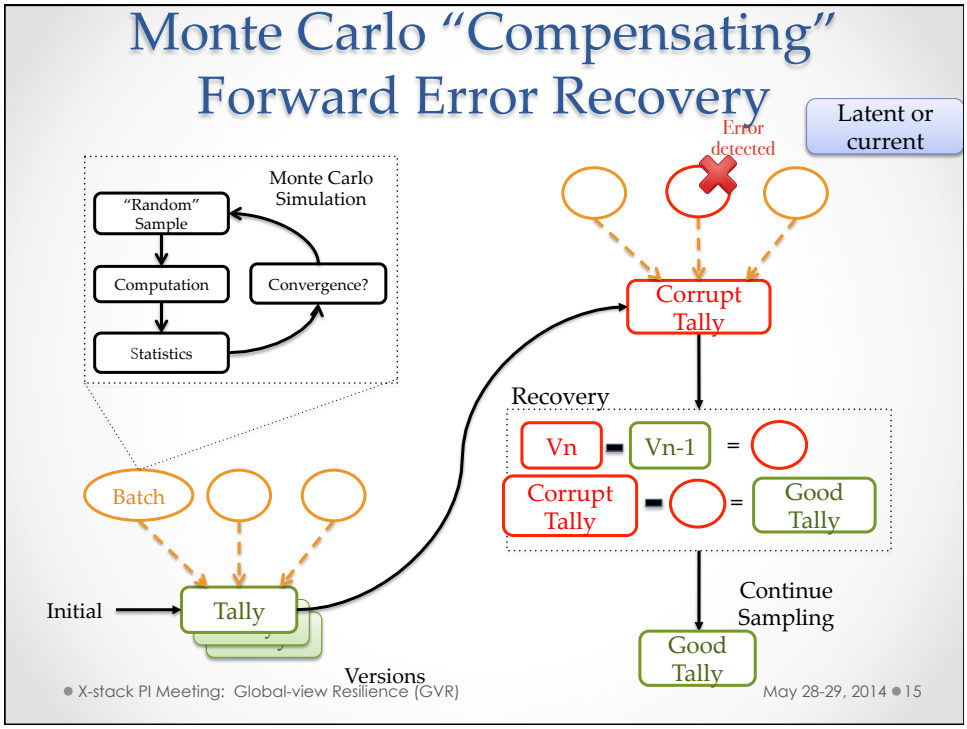
- High fidelity, computation intensive and large memory (100GB~ cross sections and 1TB~ tally data)
- Particle-based parallelization is used with data decomposition
- Partition tally data by global array
- OpenMC: best scaling production code
- DOE CESAR co-design center "co-design application"

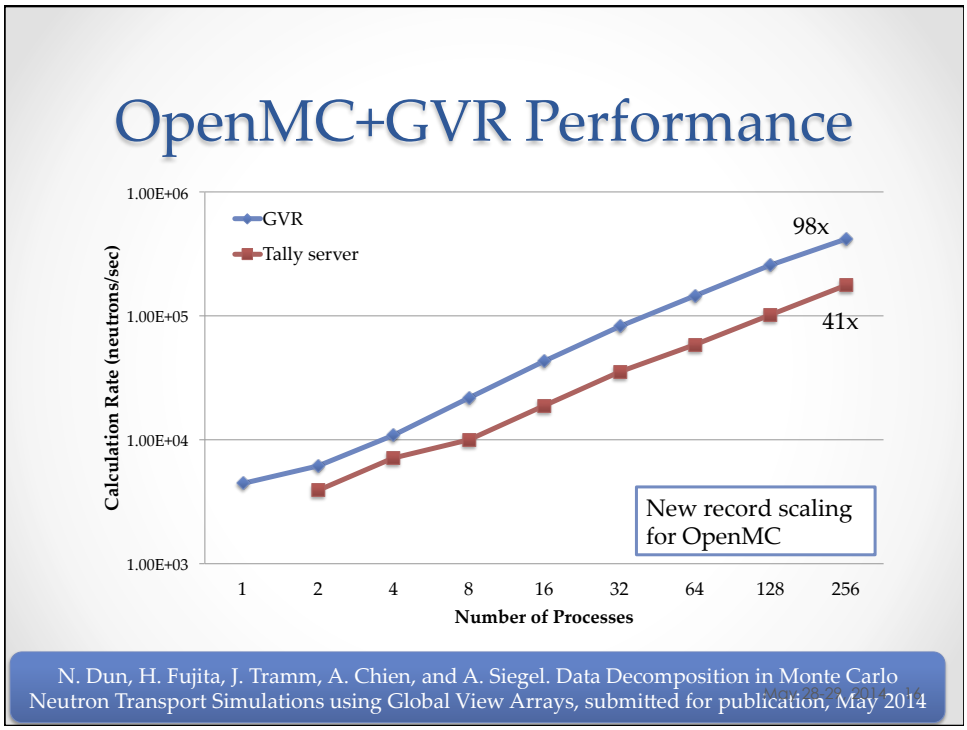
ANL/CESAR (Siegel, Tramm)

Adding Resilience to OpenMC with GVR

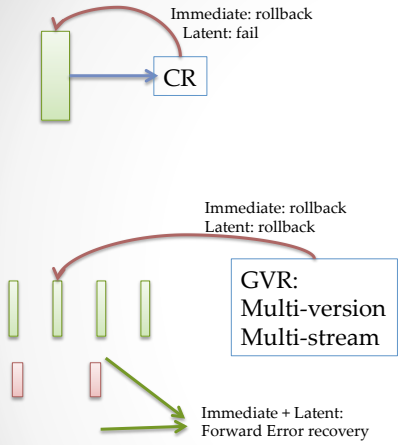
```
Initialize initial neutron positions
GDS_create(tally & source_site); //Create global tally array and source sites
for each batch
  for each particle in batch
    while (not absorbed)
        move particle and sample next interaction
        if fission
          GDS_acc(score, tally) // tally, add 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
Latent or current
Error detected
Monte Carlo Simulation
"Random" Sample
Computation
Convergence?
Statistics
Corrupt Tally
Recovery
Vn
Vn-1
=
Corrupt Tally
=
Good Tally
Batch
Initial
Tally
Versions
Continue Sampling
Good Tally
X-stack PI Meeting: Global-view Resilience (GVR)
May 28-29, 2014 15


OpenMC+GVR Performance
1.00E+06
1.00E+05
1.00E+04
1.00E+03
Calculation Rate (neutrons/sec)
GVR
Tally server
98x
41x
New record scaling for OpenMC
1
2
4
8
16
32
64
128
256
Number of Processes
N. Dun, H. Fujita, J. Tramm, A. Chien, and A. Siegel. Data Decomposition in Monte Carlo Neutron Transport Simulations using Global View Arrays, submitted for publication, May 2014

GVR enables Flexible Recovery



(Comparison to State-of-Art)

- 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's data-oriented resilience enables flexible error recovery and scaling to Exascale resilience

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

Open Resilience

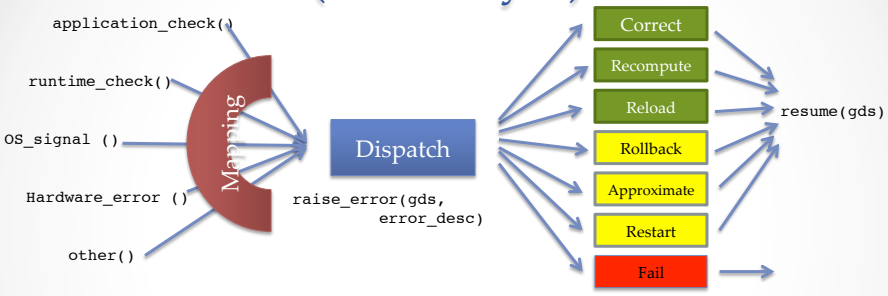
- How to maximize recoverable errors?
- How to enable recovery based on any information? (HW,OS,runtime,programming model, application, ...)

Scaling forward:

- Recover new types of errors?
- Recover in new algorithms?
- New algorithms and methods for recovery? (forward recovery)

- => Need a resilience architecture that enables and rewards investment in error recovery

Unified Signaling and Recovery (Cross-layer)



- Unified Signaling from HW, OS, Runtime, Application
  - Fail stop => raise error and expose for flexible recovery
- Application-defined error checking and error handling

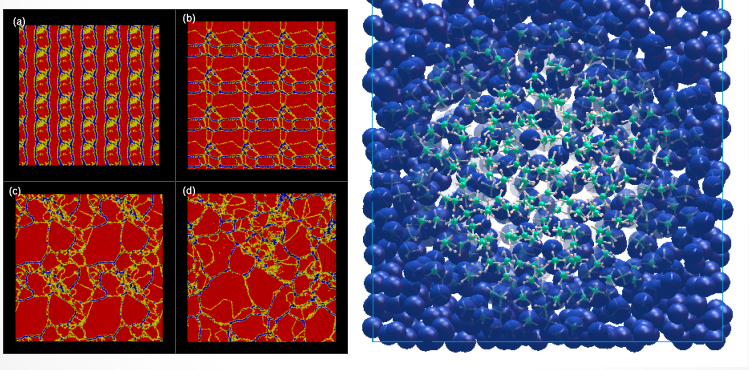
- Custom x-layer error handling
  - Prior Work: Sandia/UNM (Bridges, Brightwell,..), CIFTS/FTB (ANL, ORNL, etc.) + more...
  - Paired notification and recovery routines [Silos]
  - Exploit x-layer semantics
  - Add more as resilience challenges increase

Error Handling Generalization in GVR

- Built open resilience in GVR, started building examples
  - Discovered natural matches: 1-to-1
  - Discovered large "semantic gap" across the layers
  - How to span this gap?
- Discovered "generalization" of error recovery
  - Example 1: ddcMD
  - Example 2: Chombo
- How to maximize leverage from generalization? (create a resilience investment ecosystem)

Molecular Dynamics: miniMD, ddcMD

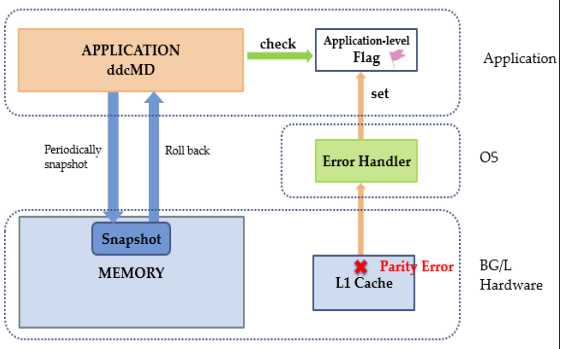
- miniMD: a SNL mini-app, a version of LAMMPS
- ddcMD is the atomistic simulation developed by LLNL -- scalable and efficient.



LLNL (Dave Richards & Ignacio Laguna)

ddcMD x-layer Error Handling (original)

```
main() {
  simulation_loop() {
    computation();
    if detects L1 cache parity error
      set flag = true;
    /* At designated rally point,
      each task check the flag */
    if rally point {
      if flag == true {
        roll back;
        continue;
      }
    }
    /* snapshot state periodically */
    if (snapshot_point)
      snapshot_state
  }
}
```



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);
}
```

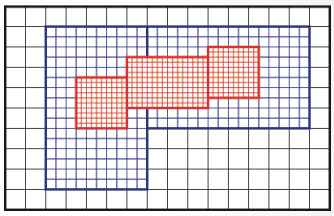
A. Fang, I. Laguna, D. Richards, and A. Chien. "Applying GVR to molecular dynamics: ..." CS TR-2014-04, Univ of Chicago.

Generalization in ddcMD

- Learn from prior x-layer experience
  - Start: BG/L L1 cache failure
  - Replicated GBelll Prize functionality (1 month of 1st year graduate student)
- GVR's Open Resilience casts error handling in a generalized error type
  - HW trap L1 error => "don't crash, set flag in user-space"; program stores "good state periodically", polls flag, and rallies
  - HW trap L1 error => "dont crash, signal data corruption using GVR"
- More checks added and grouped together
  - Application checks (various ABFT, checksum, etc.)
  - Other HW errors: DRAM, L2, L3, Interconnect, "processor check", etc.
  - Other SW errors: operating systems, communication, filesystem failures
- Result: Original L1 error recovery handler generalizes to broad range of errors
  - Errors the handler designer "never heard of"; application leverage
  - => further there are also other ways to respond... Refinement (system leverage)

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
- GVR used to version each level separately; exploits application-level snapshot-restart
  - Achieves data-corruption resilience for AMR
  - => future: customize or localize recovery
- ~ 0.7K LOC change in >500K Chombo
  - Most complex logic in saving global metadata in self-describing format

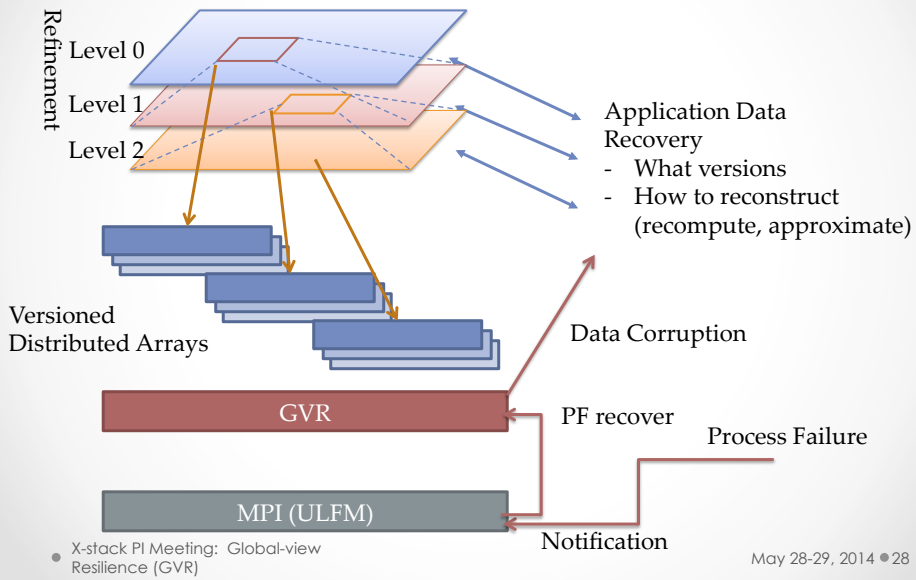


ExReDi/LBNL (Dubey, Van Straalen)

Open Resilience Insight: Generalize Data corruption recovery

- Generalization: cooperative error handling across layers (a different kind of cross-layer)
  - Start: traditional "data corruption" recovery in GVR;
  - Data error signalled by HW (memory, L1, checksum)
  - Recovers data and resumes computation (rollback, forward recovery – approximation)
  - Inspired by Dubey's prior work on ABFT forward recovery [FTXS'13]
- Idea: Can we transform a process crash in to data corruption recovery?
  - Programmer writes error recovery handler for data corruption
  - Winds up with an application that handles both data corruption and process crashes

Cross-layer AMR Recovery



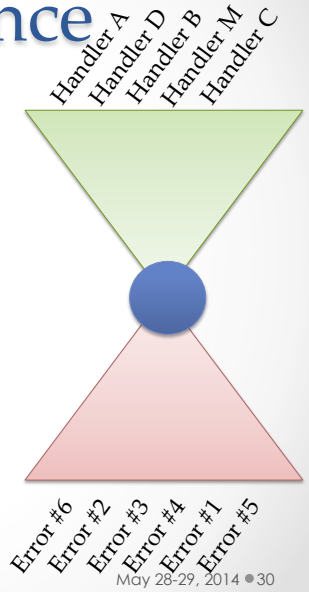
Cooperative Cross-layer Recovery

- User-level Fault Mitigation (ULFM)
  - Process crash error signaled (by system or heartbeat)
  - ULFM Detect and recover communication substrate
  - ULFM signals error through GVR Open Resilience system
- GVR Open resilience invokes Chombo's GVR data corruption recovery code
  - Detects which data collections are integral (global view distributed arrays)
  - Picks and partially materializes the needed versions
  - Recovers appropriately
- Programmer who built data corruption code gets process failure recovery "for free"
  - Resilience investment in a code "yields dividends"
- => Example of generalization... and a Resilience Eco-system

Demo in Resilience Tech Marketplace: Data corruption recovery, process fail recovery, data corruption – all with one application handler

Open Resilience

- Connects error signaling and handling across layers
  - Could coordinate resilience investment across layers
  - Complements OSR: Argo/BEACON, Hobbes/ GIB, ...
- Portable investment (application)
- Portable investment (hardware vendors)
- Can we create a resilience ecosystem?



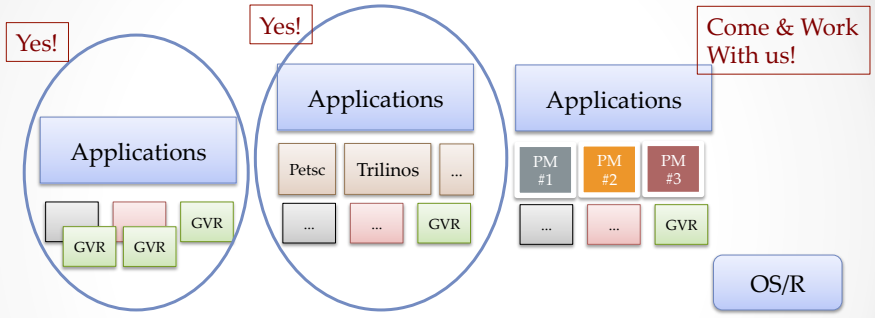
GVR Status

- Realized and Established GVR Model
  - Usable and portable today, modest code change, software architecture compatible
- Gentle slope to Exascale resilience
  - Multi-version, multi-stream model, evolution to higher error rates, forward error recovery
- GVR is application-controlled, data-oriented resilience + latent errors, forward correction
  - Contrast to CR: user-level data structures, multiple versions, multiple streams, application-controlled flexible recovery, x-layer recovery
  - Contrast to CD: whole computation, end-to-end, data not the computation, hierarchy possible, but not required, flexible forward recovery, x-layer recovery
- Path forward to x-layer resilience eco-system

Next Steps

- Preview release generally available, Sept 2014
  - Numerous partner releases already
- Broaden/continue application and library work
  - Establish and improve interface as stable and portable
  - Incorporation in applications and libraries
- Engage X-stack programming models!
- Grow Open Resilience engagement with OSR, X-stack runtime, programming model, FFwd architecture teams
- Exploit increasing opportunities in novel storage and complex memory hierarchies

GVR X-stack Synergies



- Direct Application Programming Interface
  - Co-existence, even targeted by other Runtimes
- Rich Solver Library Building Block
- Programming System Target

Call to Partnership

- GVR provides simple, portable, flexible mechanisms for resilience
- Exploiting GVR can be done with modest effort/ code, in many software structures
- Open Resilience offers opportunities for cooperative runtime, compiler, OS recovery
- Collective redundancy exposes many opportunities for optimization (and exploitation of novel storage and memory hierarchies)
- Software Availability
  - Partner release (Sept 2013)
  - Broad software release (Sept 2014)

More GVR Information

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.

Application Studies

- 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, April 2014. Submitted for publication.
- 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.

GVR Architecture and Implementation Research

- Hajime Fujita, Nan Dun, Zachary A. Rubenstein, and Andrew A. Chien. Log-structured global array for efficient multi-version snapshots. In Submitted for publication, 2014.
- 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.

Acknowledgements

- GVR Team: Hajime Fujita, Zachary Rubenstein, Guoming Lu (UC->UESTC), Aiman Fang, Ziming Zheng (UC), Pavan Balaji, James Dinan (Argonne->Intel), Pete Beckman, Kamil Iskra, (ANL), Robert Schreiber (HP), and application partners Andrew Siegel (Argonne/CESAR), Jeff Hammond (Argonne/ALCF/NWChem), Mike Heroux and Mark Hoemmen (Sandia), Dave Richards (LLNL), Anshu Dubey and Brian Van Straalen (LBNL)
- Department of Energy, Office of Science, Advanced Scientific Computing Research DE-SC0008603 and DE-AC02-06CH11357

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