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
- => Recover "immediate" errors

Make Systems Resilient

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

(c) Andrew A. Chien
April 30, 2015
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(), GDS Resume()

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

- 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);

/* 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);

Multiple versions enable more sophisticated recovery.
Simple Version Recovery: Preconditioned Conjugate Gradient

- **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)

```
A = ...
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_{old} = \rho
7:   \rho = (r, z)
8:   \beta = \rho/\rho_{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
```
Multi-stream in PCG: Matching redundancy to need

- Low redundancy
- High redundancy
- Medium redundancy

Iteration

(c) Andrew A. Chien
Molecular Dynamics: miniMD, ddcMD

- miniMD: a SNL mini-app, a version of LAMMPS
- ddcMD is the atomistic simulation developed by LLNL -- scalable and efficient.
```c
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);
}
```
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”

(c) Andrew A. Chien

ANL/CESAR (Siegel, Tramm)
OpenMC + GVR

Initialize initial neutron positions

\texttt{GDS\_create(tally \& source\_site); //Create global tally array and source sites}

\textbf{for each} batch

\textbf{for each} particle in batch

\textbf{while} (not absorbed)

move particle and sample next interaction

\textbf{if} fission

\texttt{GDS\_acc(score, tally) // tally, add score asynchronously}

add new source sites

\textbf{end}

\texttt{GDS\_fence() // Synchronize outstanding operations}

resample source sites \& estimate eigenvalue

\textbf{if} \texttt{(take\_version)} \texttt{GDS\_ver\_inc(tally) // Increment version}

\texttt{GDS\_ver\_inc(source\_site) // Increment version}

\textbf{end}

\textbf{end}

\begin{itemize}
  \item Create Global view tallies
  \item Versioning: 259 LOC (<1%)
  \item Forward recovery: 250 (<1%)
  \item Overall application: 30 KLOC
\end{itemize}
Monte Carlo “Compensating” Forward Error Recovery

Monte Carlo Simulation

"Random" Sample

Computation

Statistics

Convergence?

Batch

Initial

Tally

Latent or current

Error detected

Corrupt Tally

Recovery

Vn = Vn-1

Corrupt Tally

Continue Sampling

Good Tally

(c) Andrew A. Chien

April 30, 2015
OpenMC+GVR Performance

New record scaling for OpenMC!!

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

(c) Andrew A. Chien

ExReDi/LBNL (Dubey, Van Straalen)
## GVR Gentle Slope

<table>
<thead>
<tr>
<th>Code/Application</th>
<th>Size (LOC)</th>
<th>Changed (LOC)</th>
<th>Leverage Global View</th>
<th>Change SW architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trilinos/PCG</td>
<td>300K</td>
<td>&lt;1%</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Trilinos/Flexible GMRES</td>
<td>300K</td>
<td>&lt;1%</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>OpenMC</td>
<td>30K</td>
<td>&lt;2%</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ddcMD</td>
<td>110K</td>
<td>&lt;0.3%</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Chombo</td>
<td>500K</td>
<td>&lt;1%</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

GVR enables a gentle slope to Exascale resilience
GVR Performance (Overhead)

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

(c) Andrew A. Chien

April 30, 2015
GVR Recovery Cost
(Versions)

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

(c) Andrew A. Chien
April 30, 2015
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

Error Generation Error Detection
Running Error Latent Error Detected

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.
Access to these arrays is provided through dedicated library versioning and control timing and frequency (multi-stream). To use and portable, enabling convenient portable resilience.

Data for errors, and to recover from said errors (application-

A. Global View Resilience

Specific contributions include:
- Traditional flat array approach using several micro-benchmarks evaluate and compare the log-structured approach to the traditional flat array approach. The results show that log-structured arrays can achieve low latency and high bandwidth further, in systems with RMA, log-structured implementations can achieve low latency and high bandwidth.
- Adding NVRAM or SSD to the system resources, versioning runtime overheads can be negligible (within 74%), but as expected incur additional overheads.
- In short, overall evaluation of versioning in flat (traditional) and log-structured implementations using a variety of micro-benchmarks.

B. Operational semantics

C. Skewed and Multi-version in-memory representations

OS page tracking, dirty bits, SW declared

III. D.

Processes

Put

Get

Versions

Tail pointer Metadata Data

Initial Data Version 0 Version 1

Log head Log tail

Tail pointer of the log.

Comparative Studies with applications + varied memory hierarchies


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)
  o Tested with Miniapps – miniMD, miniFE experiments, and Full apps – ddcMD, PCG, OpenMC, Chombo

• Features
  o Versioned distributed arrays with global naming (a portable abstraction)
  o Independent array versioning (each at its own pace)
  o Reliable storage of the versioned arrays in memory, local disk/ssd, or global file system (thanks to Adam and SCR team!)
    o Whole version navigation and efficient restoration
    o Partial version efficient restoration (partial “materialization”)
    o C native APIs and Fortran bindings
    o Runs on IBM Blue Gene, Cray XC, and Linux Clusters

• Key: all of the application investment is portable because the abstractions are portable

(c) Andrew A. Chien  April 30, 2015
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 Architecture and Implementation Research

• Hajime Fujita, Kamil Iskra, Pavan Balaji, and Andrew A. Chien, "Empirical Characterization of Versioning Architectures", submitted for publication.
More GVR Info II

Application Studies


(c) Andrew A. Chien
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 Hoемmen, 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/](http://gvr.cs.uchicago.edu/)