Versioned Distributed Arrays for Resilience in Scientific Applications: Global View Resilience

Andrew A. Chien, The University of Chicago and Argonne National Laboratory
ICCS 2015, June 1
Reykjavik, Iceland
Outline

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

Global-view Data
Data-oriented Resilience

Applications

System

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

(c) Andrew A. Chien
June 1, 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
  o New, federation interfaces
  o GDS_alloc(...), GDS_create(...)

• Global view Data access
  o Data: GDS_put(), GDS_get()
  o Consistency: GDS_fence(), GDS_wait(),...
  o Accumulate: GDS_acc(), GDS_get_acc(), GDS_compare_and_swap()

• Versioning
  o Create: GDS_version_inc(), Navigate: GDS_get_version_number(),
    GDS_move_to_newest(), ... 

• Error handling
  o Application checking, signaling, correction: GDS_raise_error(),
    GDS_register_local_error_handler(),
    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.
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)

```plaintext
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
```

Unlike many other methods, CG functions only for symmetric matrices. The symmetry of the matrix is used to simplify the algorithm. In a general Krylov subspace method, we need to keep track of the entirety of the subspace over which we are currently minimizing. Due to symmetry, CG needs only to keep track three vectors of length $m$: the current approximate answer $x$, the current residual $r$, and the current direction of search $p$.

Our particular implementation also caches two iterations of the scalar $\rho = (r, r)$. Note that $r$ is updated in-place, rather than being recalculated in each iteration from $b - Ax$. This means that, if a fault occurs in the computation, the values of $r$ and $b - Ax$ may diverge.

The norm residual $\|r\|$ for CG is expected to converge at an exponential rate. In general, each iteration of $\|r\|$ should be smaller than the previous iteration by some factor. The convergence factor is dependent on the spectral condition number of $A$ [50, p. 215].
Multi-stream in PCG: Matching redundancy to need

Iteration

A

x

p

0

1

2

3

4

5

6

0

1

2

3

4

5

6

Low redundancy

High redundancy

Medium redundancy

(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) Andrew A. Chien

LLNL (Dave Richards & Ignacio Laguna)
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)
Initialize initial neutron positions

\[ \text{GDS\_create(tally \& source\_site);} \quad \text{//Create global tally array and source sites} \]

\textbf{for each} batch

\textbf{for each} particle in batch

\textbf{while} (not absorbed)

move particle and sample next interaction

\textbf{if} fission

\[ \text{GDS\_acc(score, tally);} \quad \text{// tally, add score asynchronously} \]

add new source sites

\textbf{end}

\[ \text{GDS\_fence();} \quad \text{// Synchronize outstanding operations} \]

resample source sites & estimate eigenvalue

\textbf{if} (take\_version) \[ \text{GDS\_ver\_inc(tally);} \quad \text{// Increment version} \]

\[ \text{GDS\_ver\_inc(source\_site);} \quad \text{// Increment version} \]

\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
  - Convergence?
  - Statistics

- Initial Tally
- Batch

- Recovery
  - V_n = V_{n-1}
  - Corrupt Tally
  - Good Tally

- Error detected
- Latent or current

- Convergence?
- Continue Sampling

- Versions
- (c) Andrew A. Chien

June 1, 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%.

GVR performance scales over versions and partial materialization too!
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
Additional GVR Research
Latent Error Recovery

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

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.


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

June 1, 2015
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", in CLUSTER, October 2015.
• Hajime Fujita, Nan Dun, Zachary A. Rubenstein, and Andrew A. Chien. Log-structured global array for efficient multi-version snapshots. In CCGrid 2015..

(c) Andrew A. Chien
Application Studies


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/