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

Hajime Fujita*†, Nan Dun*†, Aiman Fang*, Zachary A. Rubenstein*, Ziming Zheng‡,
Kamil Iskra†, Jeff Hammonds§, Anshu Dubey¶, Pavan Balaji†, Andrew A. Chien*†
*University of Chicago †Argonne National Laboratory
‡HP Vertica §Intel Labs ¶Lawrence Berkeley National Laboratory

I. Introduction

Resilience is a great challenge for future exascale computers. In such a hardware, dealing with all the failures in hardware level could be almost impossible or inefficient. Moreover, there are various kinds of errors, many of them cannot be handled by existing approaches, such as latent errors (or often called silent data corruption).

We propose Global View Resilience (GVR), a new library that exploits a global view data model and adds reliability through versioning (multi-version), user control timing and rate (multi-stream), and flexible cross layer error signaling and recovery. GVR enables programmers to exploit deep scientific and application code insights to manage resilience and its overhead in a flexible, portable fashion.

II. Global View Resilience

GVR provides several primitives to assist reliable execution of applications. It provides multi-version, multi-stream distributed array as a reliable storage, and also unified error handling interface for leveraging the investment to application-level error handling mechanisms. GVR also enables data-oriented resilience, protecting application data as a foundation of resilience. An application programmer specifies which data to protect, how much cost to spend on protecting it.

GVR is designed to allow incremental investment on resilience for existing scientific applications. Therefore, we implement GVR as a user-level library on top of MPI-3, such that an application programmer can easily insert library calls to where the program requires resilience. This means changes required for existing code are minimal.

A. Multi-version, Multi-stream Distributed Array

GVR provides a PGAS-style distributed array similar to GA [1], reliable data store as a foundation of resilient execution. There are two unique ideas in GVR arrays. One is the concept of multi-version array (Fig. 1). Applications control the timing and rate to create a version. And GVR intelligently maintains an appropriate number of versions across storage hierarchy subject to error rate, capacity limitation, etc. When a recovery is needed, applications can retrieve an arbitrary version preserved in GVR. We define preserved versions as read-only, which simplifies data movement, size optimization, hardening, etc. Multi-version scheme enables applications to employ powerful recovery schemes from complex errors such as latent errors [2]. Under assumption of latent errors, there is a latency between the actual error occurrence and detection. Thus not only the current contents of the array, but also several recent versions may have

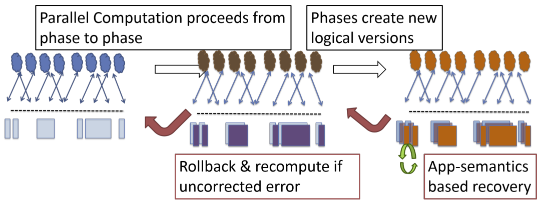
Parallel Computation proceeds from phase to phase
Phases create new logical versions
Rollback & recompute if uncorrected error
App-semantics based recovery

Fig. 1. Multi-version distributed array in GVR preserves multiple snapshots as computation evolves.

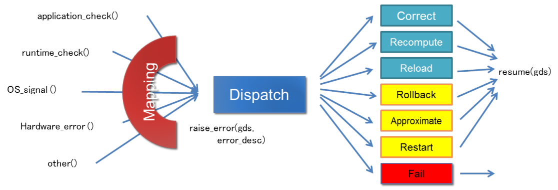

Fig. 2. Unified error handling interface allows application programmers to write error handlers for different kinds of errors in the same style.

been already corrupted by the error. In that case the application can reload some old version which is not affected by the error. With traditional checkpoint/restart systems, this is impossible because they provide access only to the newest checkpoint which might have been corrupted. The second concept is multi-stream. Applications can create multiple arrays, each is independently managed. Because of this property different arrays can be optimized for different resilience requirement. For example, read-only or read-mostly array does not need frequent snapshot, whereas heavily modified array should be versioned more frequently.

B. Open Resilience

Program execution can encounter varieties of errors, such as node crash, memory error, network error, sanity check error, etc. GVR aims to allow applications to handle all kinds of errors across all kinds of layers, in order to maximize the recoverable errors. However, this would require more programmer effort for writing error handlers. To mitigate the cost, GVR provides a unified error signaling & handling interface (Fig. 2), which provides flexible matching between error signals and error handlers. Thus one error handler could also be generalized for other kinds of errors.

TABLE I. REQUIRED CODE CHANGES FOR APPLYING GVR

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

III. APPLICATION STUDIES

To explore the utility of GVR with HPC applications, we have done studies using GVR's multi-versioning approach to add resilience to a number of applications, miniMD, ddcMD, miniFE, PCG with Trilinos, GMRES with Trilinos, Chombo, etc. Here we discuss the following four applications.

ddcMD[3]: We replicated L1 cache parity error recovery capability in ddcMD and generalize the classes of errors that can be detected and recovered, including memory system errors (L2, L3, DRAM), hardware corruption, network error, etc. GVR's distributed arrays match distributed particle and velocity structures gracefully. Versions create temporal redundancy well suited to error recovery in particle simulation.

PCG[4]: We build GVR-provided resilience into linear algebra primitives rather than requiring the applications to interact with GVR directly, via Trilinos vector objects with methods to snapshot and restore states on demand with GVR.

OpenMC[5]: We made OpenMC resilient by data versioning to the tally data which is region-based and accumulated data. GVR enables efficient forward correction for OpenMC.

Chombo[6]:GVR's multi-stream array matches the data hierarchy structure in AMR well, where data with different resolution evolves in different time step resolution.

IV. EVALUATIONS

A. Code Changes

Table I shows the sizes of code changes when applying GVR to applications. From the table we see that the required code change is less than 2%. Also, no software architecture changes are required for existing applications. These results suggest that GVR can be easily applied to existing applications.

B. Runtime Performance Overhead

We measured the runtime overhead of GVR. Experiments for ddcMD and OpenMC were done on the UChicago RCC Midway, whereas experiments for Chombo were conducted on NERSC Edison. As for the MPI library, we used MVAPICH2-2.0 Midway and Cray MPT 7.0.0 on Edison.

Tables II show the results for three applications. "Native" means the original application without GVR, "GVR" means GVR-augmented but performs no versioning, and "GVR Nm" means the application creates a version every N minutes. Numbers shown are the elapsed time in seconds, and the percentages are the relative overhead compared to the native run (for OpenMC baseline is the GVR-augmented version without versioning). The results show that the overhead is less than 5%. For versioning every 30 minutes, which is reasonable recovery in actual failure rates, the overhead is less than 1%.

TABLE II. RUNTIME OVERHEAD FOR APPLICATIONS

ddcMD

#procs	8	64	256	512
Native	1807 (0.00%)	2135 (0.00%)	1862 (0.00%)	1897 (0.00%)
GVR	1810 (0.21%)	2111 (-1.13%)	1840 (-1.23%)	1906 (0.45%)
GVR 30m	1801 (-0.29%)	2003 (-6.20%)	1836 (-1.42%)	1910 (0.69%)
GVR 15m	1804 (-0.14%)	2010 (-5.86%)	1844 (-0.98%)	1924 (1.44%)
GVR 5m	1808 (0.06%)	2034 (-4.73%)	1832 (-1.63%)	1985 (4.62%)

	#procs	8	64	256
OpenMC	GVR	3302 (0.00%)	4687 (0.00%)	6253 (0.00%)
	GVR 30m	3315 (0.40%)	4677 (-0.21%)	6270 (0.27%)
	GVR 15m	3297 (-0.14%)	4725 (0.80%)	6258 (0.08%)
	GVR 5m	3286 (-0.48%)	4704 (0.35%)	6287 (0.55%)

	#procs	128	256	1024
Chombo	Native	1746 (0.00%)	1747 (0.00%)	1726 (0.00%)
	GVR	1757 (0.64%)	1750 (0.22%)	1733 (0.38%)
	GVR 30m	1756 (0.59%)	1752 (0.29%)	1742 (0.89%)
	GVR 5m	1768 (1.30%)	1784 (2.12%)	1782 (3.24%)

V. SUMMARY AND FUTURE WORK

GVR enables portable, flexible application controlled resilience. It requires minimal change to existing applications and incurs negligible performance overhead. In future, we will continue to improve the implementation and work with community to establish Open Resilience APIs, infrastructures and portable error types/handling.

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. This work was supported by the Office of Advanced Scientific Computer Research, Office of Science, U.S. Department of Energy, under Award DE-SC0008603 and Contract DE-AC02-06CH11357 and completed in part with resources provided by the University of Chicago RCC, and resources of the NERSC, 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.

REFERENCES

[1] J. Nieplocha, B. Palmer, V. Tipparaju, M. Krishnan, H. Trease, and E. Aprà, "Advances, applications and performance of the Global Arrays shared memory programming toolkit," *International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 203–231, Summer 2006.

[2] G. Lu, Z. Zheng, and A. A. Chien, "When is multi-version checkpointing needed?" in *Proceedings of the 3rd Workshop on Fault-tolerance for HPC at extreme scale*, ser. FTXS '13. New York, NY, USA: ACM, 2013, pp. 49–56.

[3] F. H. Streitz, J. N. Glosli, M. V. Patel, B. Chan, R. K. Yates, B. R. de Supinski, J. Sexton, and J. A. Gunnels, "Simulating solidification in metals at high pressure: The Drive to Petascale Computing," *Journal of Physics: Conference Series*, vol. 46, no. 1, pp. 254–267, 2006.

[4] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps *et al.*, "An overview of the trilinos project," *ACM Transactions on Mathematical Software (TOMS)*, vol. 31, no. 3, pp. 397–423, 2005.

[5] P. K. Romano and B. Forget, "The OpenMC Monte Carlo particle transport code," *Ann. Nucl. Energy*, vol. 51, pp. 274–281, 2013.

[6] P. Colella, D. Graves, N. Keen, T. Ligocki, D. Martin, P. McCorquodale, D. Modiano, P. Schwartz, T. Sternberg, and B. Van Straalen, "Chombo software package for AMR applications design document," Lawrence Berkely National Laboratory, Applied Numerical Algorithms Group, Computational Research Division, Tech. Rep., 2009.