Applying GVR to Molecular Dynamics: Enabling Resilience for Scientific Computations

Aiman Fang and Andrew A. Chien
{aimanf,achien}@cs.uchicago.edu

Department of Computer Science
The University of Chicago
Chicago, IL 60637

Math & Computer Science Division
Argonne National Laboratory
Argonne, IL

Abstract

In future exascale systems [1]-[4], resilience is a major concern. Molecular dynamics codes are an important computational method in a wide variety of areas of biology, chemistry, and physics. We applied the GVR (global view resilience) library to the ddcMD (domain decomposition molecular dynamics) code, both to explore application resilience challenges and evaluate the potential for GVR to broaden and simplify application resilience.

Following the ddcMD code changes made to tolerate hardware unrecoverable L1 cache parity errors [17], we replicated these recovery capabilities with only adding 310 lines of GVR library calls to original 10,935 lines of source code. Our next step was to use this base to explore a range of application-specific error detection and recovery schemes that generalize the classes of errors that can be detected and recovered without application interruption. This broader class of errors includes general memory system errors (L2, L3, DRAM, bus, controller, etc), hardware computation errors, communication errors, software bugs, and others. The error checks are conveniently expressed in the application source code in terms of application data structures, and enable flexible, application-controlled recovery from these errors. We find that GVR enables convenient broadening of error coverage and resilience.

To evaluate the capabilities of error detection schemes, we performed error injection experiments. The results show that application-specific error detection schemes can detect certain magnitudes of errors, but leave some errors silent. Our GVR provides opportunities to recover from silent errors.

Keywords: resilience, fault-tolerance, high performance computing, large scale computing, molecular dynamics

1. INTRODUCTION

Scientific and engineering computations have driven the demand for large-scale computing. Today's top high performance systems have achieved peak performance of 10^{15} floating point operations per second and they are continuing to progress towards 10^{18} floating point operations per second for exascale systems. These large-scale systems are comprised of millions of components, leading to higher error rates. It is anticipated that the mean time between failures (MTBF) could be less than an hour for top parallel computers [2], [3], [4]. Consequently, resilience becomes a major concern for scientific applications running on large-scale systems.

The current broadly used method to achieve resilience is checkpoint/restart scheme. In this scheme, each process periodically snapshots its memory to keep a copy of the historical computation states. Upon an error detected, the application restarts from the latest snapshot. However, this scheme will come across several challenges when the applications and systems continue to progress to petascale/exascale. First, the time needed for taking checkpoints would be too long, which may even exceed the computation time. The large overhead cannot be acceptable. Second, this scheme will not work when the interval of checkpointing exceeds the MTBF. Third, it has limitations in tolerating varieties of errors.

The errors occurring on systems are categorized

into three types [5]: 1) hard errors, are permanent, unrecoverable errors such as hardware component failure, which require human intervention; 2) soft or transient errors, are errors that are detected and can be corrected or masked; 3) silent errors or latent errors, are errors that are not detected or only exposed after a long period. The soft errors and silent errors can lead to computation failures or invalid results, causing severe issues. For rigorous scientific applications, it is of importance to have the ability to tolerate silent errors in order to reach solid conclusions. However, the current checkpoint/restart scheme lacks the ability to tolerate these errors.

In this paper, we focus on resilience to soft errors and latent errors. We choose ddcMD molecular dynamics for this study. Following the ddcMD code changes made to tolerate hardware unrecoverable L1 cache parity errors [17], we replicated these recovery capabilities by adding 310 lines of GVR library calls. We then designed application specific error detection and recovery schemes, using the data structures and molecular dynamics properties in original source code. We further conducted error injection experiments to 1) demonstrate the resilience enabled by GVR towards various errors, 2) study the capabilities of error checks in terms of error precision.

The specific contributions of our study include:

- GVR can be applied to large-scale applications with modest code change. We only add (310 lines/10,935 lines=) 2.8% of code into ddcMD.
- The GVR model naturally broadens the class of errors that can be detected and recovered, increasing and simplifying fault tolerance.
- Application can conveniently control error detection and recovery strategies exploiting application semantics.
- GVR enables opportunities to recover from silent errors.

This paper is organized as follows: In Section 2 we introduce the background of molecular dynamics, GVR library and model; In Section 3 we describe our approach of applying GVR to molecular dynamics to achieve resilience; In Section 4 we present our error injection experiments and results; In Section 5 we discuss the results and discuss the possibilities of using GVR to recover from silent errors; Finally, in Section 6 we present our conclusions and possible future studies.

2. Background

2.1 Molecular Dynamics

MD simulation is a principal tool in broad areas of biology, chemistry and physics, offering insights concerning natural and experimental phenomena.

MD simulates the physical movements of a collection of atoms, which are allowed to interact for a period of time according to the laws of classical physics [6], [7]. The MD computation is performed in a series of time steps. Briefly, each time step consists of two phases: *force calculation* and *integration*. In *force calculation phase*, MD calculates the force on each atom according to physical laws such as Newton's equations. *Integration* phase updates the parameters of the atoms such as position and velocity. The algorithms used are much more complicated in practical. Detailed algorithms in MD can be found in a lot of previous works [8], [9], [10], [11].

MD simulation involves calculation on a large number of atoms, which inherently requires parallel computing on large-scale systems. There are many established MD codes, including CHARMM, NAMD, LAMMPS, ddcMD, varied in algorithms. Among these MD applications, ddcMD which is developed by Lawrence Livermore National Laboratory (LLNL), was built to achieve strong scaling and efficiency. In 2005, the ddcMD team of LLNL and IBM investigated the solidification in metal systems using ddcMD code on the IBM BlueGene/L computer at LLNL [12]. The size of atoms varied from 64,000 to 524,288,000. DdcMD achieved performance rates up to 103 TFlops, with a sustained rate of 101.7 TFlops over a 7 hour run on 131,072 processors [13]. The team was awarded the 2005 Gordon Bell Prize for this accomplishment [14]. In this paper, we take ddcMD as a representative of MD applications for resilience study without losing generality. It is easy to extend the approaches applied in this paper to other MD applications to obtain resilience.

Besides the scalability, ddcMD was also designed

with resilience scheme to tolerate the primary failure mode on BG/L, the transient parity errors on the L1 cache line. We describe ddcMD's default resilience scheme and the overall algorithm as following.

1) ddcMD Resilience Scheme

The primary failure mode on BG/L has been transient parity errors on the L1 cache line [13]. The L1 cache can detect single bit errors but cannot correct them. ddcMD employs a checkpoint/rollback scheme and utilizes application-level error recovery strategy. It periodically takes a fast checkpointing of the full computation state in memory, including states like positions, velocities of the atoms. The data structure of the full state is shown in figure 1. When the compute node kernel detects an unrecoverable parity error, the error handler sets a global flag. The application continues execution until it reaches a designated rally point, at which all tasks check the error flag and discard the current results, then restore to the previous backup state. This simple approach leverages the application specifics to reduce the overhead of process level checkpointing and gains fault tolerance to L1 cache parity error. We will show in later sessions, our GVR approach enables resilience towards a broader class of errors beyond L1 cache parity error.

```
typedef struct stateBackup_st {
    double _time;
    int _loop;  // number of current loop
    int _nlocal; // number of local atoms
    SIMULATE * _simulate;
    THREE_MATRIX _h;
    gid_type * _label;
    int* _atype;
    SPECIES** _species;
    GROUP ** _group;
    double* _rx;  // position x axis
    double* _ry;  // position y axis
    double* _rz;  // position z axis
    double* _vx;  // velocity x axis
    double* _vy;  // velocity y axis
    double* _vz;  // velocity z axis
    DOMAINX* _domainx;  // domain centers
} StateBackup;
```

Figure 1. The data structure of the backup state in ddcMD. ddcMD keeps one copy of this backup data structure in memory for recovery.

2) ddcMD algorithm

The computation algorithm in ddcMD is simply the numerical integration of equations of movements. A set of atoms are interacting via potential energy function [15]. The innovative approach of ddcMD to achieve high performance in parallel large-scale systems is domain decomposition. This method divides the simulation volume into domains, each assigned to a task. ddcMD can also control load imbalance via domain reassignment. Figure 2 shows the pseudo-code of the main algorithm used in ddcMD

```
main() {
    Init()  // Initialization
    Assignment() // domain decomposition
    simulation loop { // main computation
        UpdateForces()  // calculate force
        UpdateGlobal()  // update parameters
        if(parityError()) restoreState()
        else saveState()
        if(imbalance) reassignment()
    }
}
```

Figure 2. Pseudo-code of main algorithm in ddcMD

2.2 Global View Resilience (GVR)

GVR ([16], [17]) project is collaboratively developed by our group at University of Chicago and folks at Argonne National Laboratory. GVR is a new programming approach that employs a global view data model. The global view comes from the globally addressable distributed data arrays, called GDS objects. Applications create GDS objects which are used to store the essential data for computation. GVR then protects the GDS objects by creating versions (snapshots) of these global arrays. The versioning rate for each global array is specified by the applications. In a simple recovery model, upon an error detected, applications restore a previous good version of data and restart the computation. There are several key novel features in GVR that distinguish it from other resilience schemes.

1) Data-oriented: GVR builds the computational reliability on a foundation of data reliability. Computation is viewed as a series of transformations from one global data state to the next global data state.

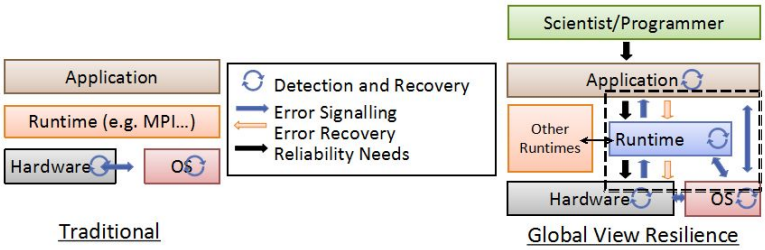

Figure 3. Traditional resilience model and GVR model. The traditional model restricts the error detection and recovery in hardware and OS, thus lacks the resilience capabilities. GVR model enables rich error checking and recovery at each layer (application, runtime, hardware, OS). Applications benefit from the cross-layer error management in flexibility.

GVR allows applications to express the resilience at the granularity of data array. Applications create GDS objects (global distributed arrays) to store important data structures and preserve the versions (snapshots) of GDS objects instead of the whole memory. Applications can indicate reliability priorities in terms of data structure versioning rate. Thus, the application programmers are able to control the overheads.

2) Multi-version: GDS objects or global arrays are versioned in the runtime. Instead of keeping one latest copy of the snapshot like traditional checkpoint/restart scheme, GVR preserves multiple versions of data arrays. The multi-version mechanism of GVR provides opportunities to recover from silent/latent errors. Suppose there exist silent/latent errors, the latest snapshot is corrupted, which cannot be used for recovery. In this scenario, the previous good versions provided by GVR are useful for recovery.

3) Flexible cross-layer error management: GVR model enables flexible recovery strategies. A variety of errors can be captured and exposed to applications through the hardware layer, OS layer, and application-specific error detection method defined by programmers. When an error is detected, applications can employ the optimal recovery strategy under the consideration of overheads, performance, etc. Figure 3 illustrates the difference between traditional resilience model and GVR model, showing the flexibility of cross-layer error management.

3. METHODOLOGY

To utilize the GVR model, applications first identify the important data structures that need protection. Then applications create GDS objects to store the data and create versions of the objects at specific rates. Second, programmers design application specific error detection (checking) and recovery schemes, and fit them into the applications. We describe how we follow this routine and apply GVR to ddcMD in following sessions.

3.1 Applying GVR to ddcMD

ddcMD's default resilience scheme has already identified a series of data variables that need to backup, as shown in Figure 1. We create a GDS object for each variable in the "backup" data structure, and create new versions of each GDS object every 10 time steps. We plug the error detection schemes described in *3.2* into ddcMD. In recovery, the application tries to find a correct previous version of GDS objects, and restore the data. The GVR-augmented ddcMD pseudo code is shown in figure 4.

3.2 Error detection schemes for ddcMD

Application-level error detection schemes rely on the inherent properties. For ddcMD, we identify two properties and utilize them as error detectors.

1) Atom Position Detection

As described in 2.1, ddcMD employs domain decomposition algorithm to parallelize the calculation. Each atom is assigned to a domain according to the cutoff distance range. Based on this knowledge, we can check the position of each atom to detect potential errors that cause atoms to jump out of the assigned domain. The scheme is simple without complicate calculation. And it captures any source of errors that result in atoms jumping out of range.

```
main() {
    Init()  // Initialization
    GDS_Init() // Initialize GDS objects
    Create_GDS_objects()
    Assignment() // domain decomposition
    simulation loop { // main computation
        UpdateForces()  // calculate force
        UpdateGlobal()  // update parameters
        if(errorDetection()) {
            find_correct_GDS_version()
            restore data
        }
        else
            create_new_GDS_version()
    }
}
```

Figure 4. GVR-augmented ddcMD pseudo code

2) Total Energy Variety Detection

One aspect of MD simulation is that the simulation system of atoms or particles becomes stable after running a period if there are no outside interrupts. For example, as one important parameter of MD simulation, the total energy of atoms shows only tiny changes between two time steps. Figure 5 illustrates the total energy trend during the simulation from time step 1 to time step 500. The total energy shows convergence after about 100 time steps. Based on this observation, we conjecture that a remarkable change of the total energy between time steps indicates errors.

This error detection scheme is efficient even for large-scale MD simulation. The application only needs to calculate the difference of total energy between time steps and checks if it exceeds the normal range or threshold.

There are considerable numbers of inherent properties in ddcMD that we can use as error detectors beyond these two. For example, one potential approach is to predict the weight center of atoms calculated by current velocity and position values, and then compare it with the simulation result. This approach consumes heavy computations, thus comes with high overheads.

In this paper, we use the first two error detection methods. There are two benefits. First, they are light weighted with trivial overheads. Second, these two approaches omit the sources that cause errors. No matter where the errors come from, whether occurring in memory (L1, L2, L3, bus, etc), network or software, these two methods are able to capture them by detecting symptoms caused by these errors. It is the application-controlled error detection and recovery enabled by GVR that broadens the resilience to various classes of errors.

4. EXPERIMENT & RESULTS

In this session, we present the error injection experiments and corresponding results. We further evaluate the performance of total energy variety error detection method by studying its sensitivity towards different granularities of errors.

4.1 Error-Injection Experiment

In this experiment, the size of atoms configured in ddcMD is 2000. The simulation runs for 3000 time steps. The application creates a new version of GDS objects every 10 time steps. Meanwhile, the error detection interval is 10 time steps. At time step 2000, we inject errors into the position value of each atom by increasing the value by 1.

Figure 6 illustrate the total energy from time step 1990 to time step 2010. We can observe that after errors injected at time step 2000, there is a remarkable change in total energy, indicating abnormity. The application continues execution until error checking at time step 2010. The error is detected, invoking the recovery procedure. The application restores the data of time step 2000 and restarts the computation correctly.

The results prove the effectiveness of using GVR to achieve application resilience. The errors may come from any sources, such as memory errors, network errors, software bugs. As long as their effects are captured by the error checks, they will be corrected. GVR enables the resilience towards a broader class of errors beyond L1 cache parity error.

4.2 Error Detection Sensitivity Test

In *4.1* error-injection experiment, the errors injected are of magnitude 1, which incur notable changes in total energy. In this scenario, the error detection

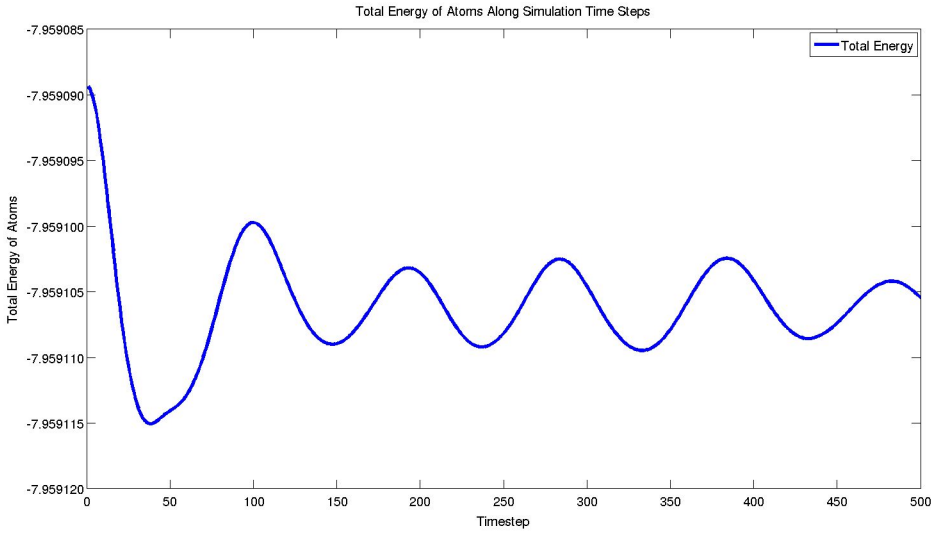

Figure 5. Total energy of atoms change along the time step. The total energy becomes relatively stable after 100 time steps. The total energy value oscillates between -7.959110 and -7.959100 and shows convergence.

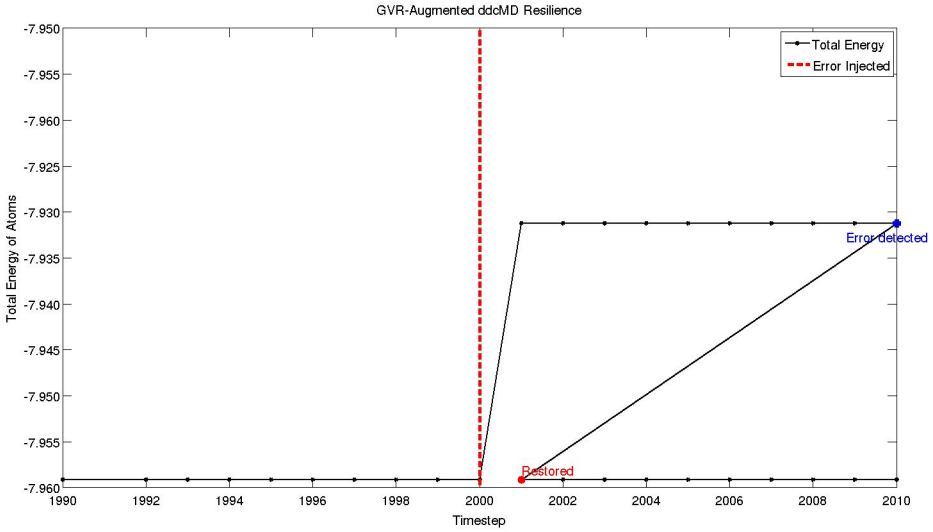

Figure 6. Total energy trend in error injection experiment. The errors are injected at time step 2000, causing the total energy jumps from -7.960 to -7.930. The application continues execution until time step 2010, the designated error checking point. The application detects the error and rolls back to time step 2000 and restarts computation. The total energy resumes at time step 2001.

method can capture the errors. In order to learn the sensitivity of total energy variety detection, we vary the granularities of injected errors. In this experiment, the size of atoms is 2000. The application runs for 1020 time steps in total. At time step 1000, we inject one error of magnitude ranging from 1e-4 to 1 into a random atom's position. The application continues to run another 20 steps. Figure 7 shows the total energy change in the following 20 steps after the error is injected at time step 1000. Without losing generality for conclusion, we repeat this experiment by injecting errors into velocity values of atoms. The results are shown in figure 8.

From figure 7, we can conclude that the total energy exhibits different granularities of changes respectively. An error of magnitude 1 incurs change of total energy at granularity 1e-3, while an error of magnitude 1e-4 causes the change at granularity 1e-7. However, the normal change of total energy between time steps in error free simulation is at granularity 1e-5. Thus, the error check cannot detect a single error occurring at position value less than 1e-3.

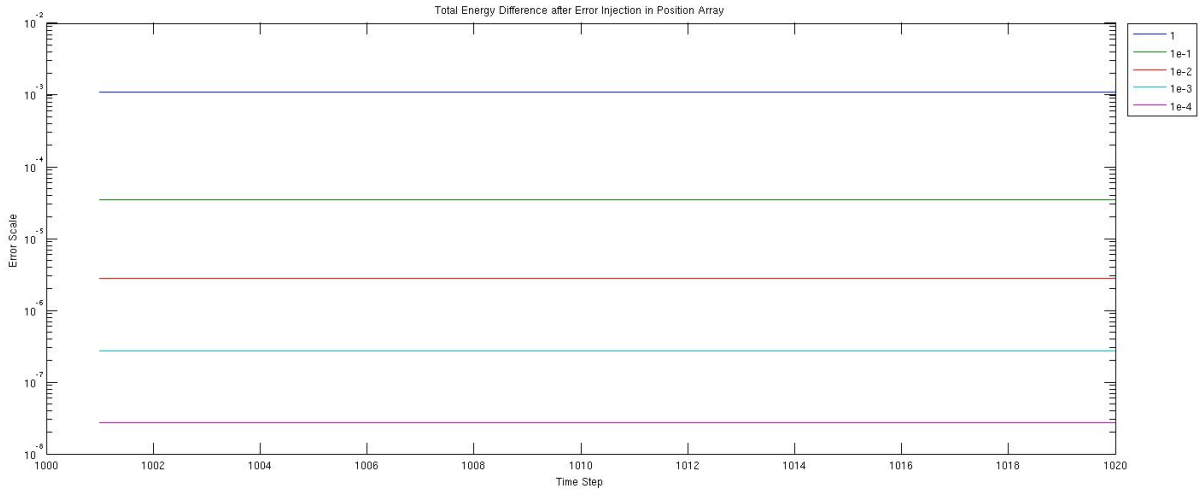

Figure 7. Total energy change in terms of various granularities of errors injected into position values of atoms.

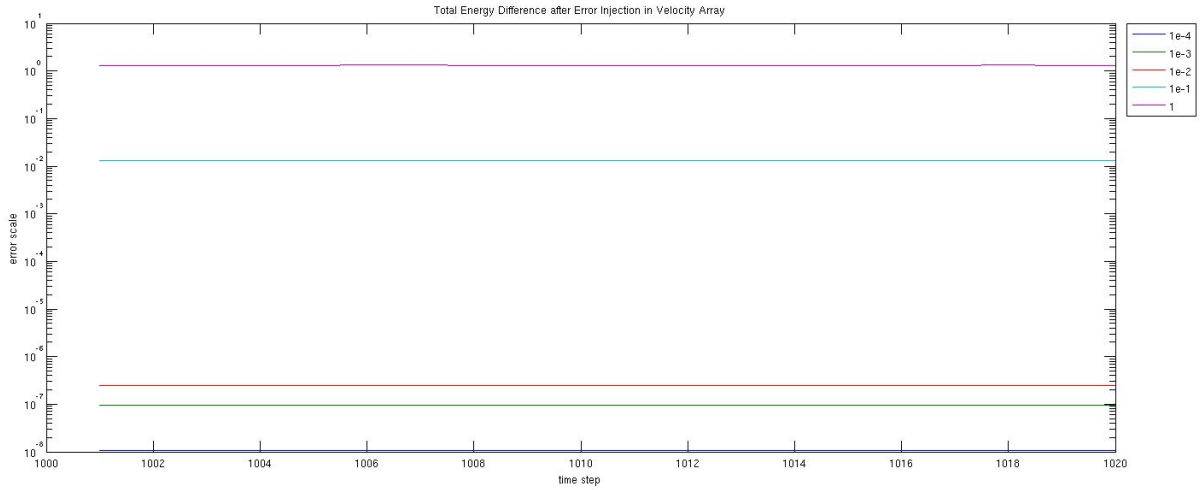

Figure 8. Total energy change in terms of various granularity of errors injected into velocity values of atoms.

Similarly, as observed in figure 8, when the size of the error is larger than 1e-2, the variety of the total energy is larger than the normal threshold in magnitude of 1e-6, therefore is detectable by the method. Below this error significance, the single error in velocity value is essentially undetectable by this method.

Furthermore, observed from figure 7 and 8, the significance of the error appears to be directly proportional to the energy change. As the simulation proceeds, the effect on energy of a single error, independent of significance, is roughly constant. So there is limit promise that time will reveal more of these errors. The undeniable limitations of the error detection method indicate silent errors are common in reality.

5. DISCUSSION

In session 4, we demonstrated that augmented with GVR, ddcMD can tolerate certain magnitudes of soft errors that are detectable by the error checking methods. The results also indicate the limitations of these error detection approaches. The large percentage of minor errors that cannot be detected will remain silent, therefore possibly result in wrong scientific conclusions.

Silent errors have been studied in [18], [19], [20]. Many existing solutions proposed to resolve silent errors rely on more sophisticated error detection methods. However, an ambitious error detection scheme usually requires heavy computations, coming with large overheads, thus can harm the performance. Besides, it is difficult to design such a method with no limitations in detecting all kinds of errors. However,

GVR provides the opportunities to tolerate silent or latent errors with its multi-version mechanism.

With GVR's multi-version mechanism, even if the latest snapshots are corrupted by silent errors that escaped error detection, there still exist multiple copies of the historical data. The application can still restore to an existing correct version of data without losing too many computation efforts. Then there come the questions of how many versions should be preserved and what the optimal versioning rates are. Paper [21] provides insights into these questions. As these parameters are configured in applications, GVR allows programmers to control the overhead and optimize the performance, given certain resilience requirements.

6. SUMMARY AND FUTURE WORK

ddcMD, as one representative of scalable molecular dynamics simulation programs, is designed with resilience to L1 cache parity error to run on parallel large-scale systems such as BG/L, BG/Q. However, it does not support fault tolerance towards other errors. In this paper, we apply GVR to ddcMD by only adding 310 lines to original 10,935 lines of source code. We demonstrate that augmented with GVR, ddcMD is able to tolerate a broader class of errors beyond L1 cache parity error in [17]. Furthermore, we design application-semantic error detection and recovery methods. Through the error injection experiments, we show the resilience capabilities of the error checks towards varied magnitudes of errors. The results indicate certain limitations of these error checks, which can possibly result in high potential of silent errors. The existence of silent errors is particularly challenging in large-scale scientific computations. We show that the multi-version mechanism of GVR can provide opportunities by allowing applications to recover with previous good versions of data without losing much computation work.

There are several interesting future study directions. One is to explore more sophisticated error detection methods. We can also design an error detection scheme that combines several error checks and leverage their costs, capabilities, and other features. Another direction rises from application-controlled error detection and recovery, which requires the trade-off between performance and resilience. There optimization problems in this area are worth exploring.

REFERENCES

[1] Saman Amarasinghe, Dan Campbell, "ExaScale Software Study: Software Challenges in Extreme Scale Systems", Sept. 14, 2009.

[2] Franck Cappello, "Fault Tolerance in Petascale /Exascale Systems: Current Knowledge, Challenges and Research Opportunities", *International Journal of High Performance Computing Applications*, Volume 23, No. 3, Fall 2009, pp. 212-226.

[3] Ricardo Bianchini, Tarek El-Ghazawi, Armando Fox, Forest Godfrey, Adolfy Hoisie, Kathryn McKinley, Rami Melhem, James Plank, Partha Ranganathan, Josh Simons, "System Resilience at Extreme Scale".

[4] Peter Kogge, Keren Bergman, Shekhar Borkar, et al, "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems".

[5] Rinku Gupta, Kamil Iskra, Kazutomo Yoshii, Pavan Balaji, Pete Beckman, "Introspective Fault Tolerance for Exascale Systems".

[6] Molecular dynamics definition, http://en.wikipedia.org/wiki/Molecular_dynamics

[7] David E. Shaw, et al, "Anton, a Special-Purpose Machine for Molecular Dynamics Simulation", ISCA '07, June 9 -13, 2007

[8] Stewart A. Adcock, J. Andrew McCammon, "Molecular Dynamics: Survey of Methods for Simulating the Activity of Proteins", *Chem Rev*. 2006 May; 106(5): 1589-1615.

[9] Kevin J. Bowers, Edmond Chow, Huafeng Xu, et al, "Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters", SC2006, Nov. 2006, Tampa, Florida, USA

[10] David Brown, Bernard Maigret, "Large Scale Molecular Dynamics Simulations using the Domain Decomposition Approach"

[11] Thierry Matthey, Trevor Cickovski, et al, "PROTOMOL, An Object-Oriented Framework for Prototyping Novel Algorithms for Molecular Dynamics", ACM Transactions on Mathematical Software, Vol. 30, No. 3, Sept. 2004, pp 237-265.

[12] Frederick H. Streitz, James N. Gloskli, Mehul V. Patel, et al, "Simulating Solidification in Metals at High Pressure: The Drive to Petascale Computing", 2006 IOP Publishing Ltd.

[13] The Need for Speed, http://www.scidacreview.org/0801/html/scale1.html.

[14] J.N. Glosli, K. J. Caspersen, J.A. Gunnels, D. F. Richards, R. E. Rudd, F. H. Streitz, "Extending

Stability Beyond CPU Millennium, A Micron-Scale Atomistic Simulation of Kelvin-Helmholtz Instability", Supercomputing '07 Nov. 2007, Reno, NV.
[15] Frank R. Graziani, et al, "Large-scale molecular dynamics simulation of dense plasmas: The Cimarron Project", High Energy Density Physics 8 (2012), pp 105-131.
[16] "Global View Resilience (GVR)", https://sites.google.com/site/uchicagolssg/lssg/research/gvr
[17] "Global View Resilience (GVR)", Argonne, http://www.mcs.anl.gov/project/gvr-global-view-resilience
[18] Austin R. Benson, Sven Schmit, Robert Schreiber, "Silent error detection in numerical time-stepping schemes"
[19] Dongarra, J., Beckman, P., Moore, T., AAerts, P., Aloisio, G., Andre, J.-C., Barkai, D., Berthou, J.-Y., Boku, T., Braunschweig, B., et al. (2011). The international exascale software project roadmap. *Intertional Journal of High Performance Computing Applications*, 25(1):3-60.
[20] Snir M., et al, "Addressing failures in exascale computing", 2013.
[21] Guming Lu, Ziming Zheng, Andrew A. Chien, "When is Multi-version checkpointing Needed?" in Proceedings of the 3rd Workshop on Fault-tolerance for HPC at extreme scale. ACM, 2013, pp. 49-56.