FROM DETECTION TO OPTIMIZATION: IMPACT OF SILENT ERRORS ON SCIENTIFIC APPLICATIONS

JON CALHOUN*, LUKE OLSON (ADVISOR)†, AND MARC SNIR (ADVISOR)‡

Abstract. As high performance computing (HPC) continues to progress, constraints on HPC system design force the handling of errors to higher levels in the software stack. Of the types of errors facing HPC, silent errors that silently corrupt system or application state are among the most troubling. Understanding how applications behave in the presence of these silent errors is critical to gain incite for effective utilization of HPC systems. This can be directed toward developing algorithmic based error detectors guided by application characteristics from error injection and error propagation studies. The realization that not every error is significant enough to warrant recovery, optimizations to consume less power at the expense of adding in small errors such as further reducing voltage or lossy compression techniques can be employed when transferring data between certain levels in the memory hierarchy.

1. Introduction and Problem Statement. High performance computing is a fundamental tool of discovery to many fields of science and engineering. Understanding limiting factors to system/application performance and usability becomes critical for maximizing the scientific impact for systems/applications. As machines grow in complexity they become more susceptible to faults [8]. The behavior of HPC systems in the presence of fail-stop faults is well understood, and checkpoint restart schemes have been effectively devised to recover from these types of failures taking advantage of the hierarchy to avoid memory bandwidth bottlenecks. However, understanding how faults that may not directly manifest as a detectable event pose a potentially limiting effects on HPC applications is less understood. These *silent errors* lead to silent data corruption (SDC), non-intended perturbation in system or application state unbeknownst to the system or application. This corrupted state can propagate, corrupting larger amounts of state from the system and/or application. In some cases, results produced from the application can be altered due to SDC; wasting system execution time and delaying collection of the intended correct results. Moreover, as Moore's Law slows, we seek new avenues for performance from current hardware by adding *errors* into the application by using lower precision or compression in computation and data storage/transfer.

Understanding how HPC applications behave in environments where SDC is common or in situations where error in purposely added into the data is needed to ensure applications run efficiently and correctly on large scale systems. When running in environments where SDC is common, HPC applications need low cost detection and recovery schemes to ensure that result altering SDC is detected and mitigated. The realization that many HPC applications can tolerate small amounts of error and still generate the correct or an acceptable result allows for optimizations to how data is transfered, operated on, and stored. Lossy compression, at the expense of adding a small controllable amount of error, can be used to dramatically reduce the size of data transfered during the application mitigating bandwidth limitations. My thesis seeks to understand how HPC applications handle SDC, how SDC can be detected at low cost, and how embracing small amounts of error can lead to new performance optimizations such as lossy compressed data movement.

2. Research Highlights.

2.1. Fault Injection. To facilitate research on SDC and its impact on HPC applications, I created the LLVM based fault injector FlipIt [1]. FlipIt instruments code as it is being compiled to allow for the modifications in the result register of instructions. Injections occur dynamically

*Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL (jccalho2@illinois.edu, http://web.engr.illinois.edu/~jccalho2/).

†Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL (lukeo@illinois.edu).

‡Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL (snir@illinois.edu).

at runtime based the current state of the application and fault injector. Although it is designed to simulate errors in combinational logic that manifest as register perturbations, many fault models can be used with FlipIt. A runtime API allows the program to dynamically modify how fault injection is carried out by selecting a fault distribution, number of injections, which MPI ranks are faulty, and etc. As the source code is compiled and when fault injection occurs, information about failure locations and other injection metrics are logged for fast and efficient off-line visualization. FlipIt is released on github[1] as an open source project and is currently used by several researchers in their personal work.

2.2. Resiliency Visualization. HPC fault injection visualization involves collection of results and statistics from thousands of runs each of which may contain thousands of processes. Many fault injection tools and techniques exists. With each having their own way of collecting results and statistics for reporting during analysis and visualization. Although each fault injector logs data specific to its operation, ultimately they record similar data for each injection and for every fault injection run of the application. With similar information being logged, there is no standard way to visualize this data. To this end, I have been working on a tool called FaultSight[2] in collaboration with an undergraduate at UIUC, Einar Horn, to visualize fault injection data. This tool originally designed for FlipIt, is now being abstracted to meet the needs of other fault injection frameworks by providing a visualization tool capable of using data from many fault injection tools. FaultSight is designed to quickly visualize properties of fault injection runs at a high level such as percent of trials completing or generating a particular signal, and when possible map these back where they occur in the source code. Moreover, the strength of FaultSight is its ability to easily select a subset of fault injection runs that meets a complex description-e.g, all bit positions that when flipped in a floating-point value resulted in a segfault on rank 4. FaultSight is still under active development and a paper on how it functions and how it can be used when designing a custom application based resiliency scheme is being prepared.

2.3. Algebraic Muligrid Resiliency. Many scientific and engineering applications rely on solving a system of linear equations. Algebraic multigrid (AMG) is an efficient $\mathcal{O}(n)$ linear solver that is often used as a preconditioner for Conjugate Gradient (CG) or Generalized Minimum Residual (GMRES). Much is known about AMG's scalability, but little research has been done to understand its behavior in a faulty environment. Research at LLNL [4] showed that AMG is naturally resilient due to its iterative nature, but is vulnerable to segfaults due to corrupted pointers.

My research on AMG resiliency [2] focuses on leveraging algorithmic properties of AMG to create a customized detection and recovery scheme. Although AMG's iterative nature makes it naturally resilient to SDC, SDC can increase the number of iterations to converge. To prevent this, two deviation checks are implemented to ensure that AMG was making sufficient progress. The first check is a heuristic based on the residual history. AMG does not guarantee that the residual decreases with every iteration, but unexpected sharp jumps in the residual can signify presence of SDC. The second check monitors the energetic stability at each level in the AMG hierarchy of linear systems to verify that it decreases with every iteration at corresponding points in the hierarchy. If the energy check is triggered, AMG is rolled back to the previous level in the AMG hierarchy to attempt re-execution. If the same detector triggers on re-execution or if the residual check is triggered, recovery proceeds from a in-memory checkpoint of the solution vector and restart the iteration. The solution to segfaults leverages the observation that segfaults occur shortly after a pointer is corrupted. AMG is built using basic linear algebra operations that operate on matrices and vectors. These operations can be written as idempotent operations. Upon detection of a segfault, AMG re-executes the idempotent basic linear algebra operation. Otherwise recovery proceeds using the hierarchical based recovery scheme. Results show the

[1] https://github.com/aperson40/FlipIt
[2] https://github.com/einarhorn/FaultSight

overhead of the detectors and recovery scheme ranges from 1%-20% depending on which detectors are enabled. A low cost, everything sans energy check, configuration of detectors has an overhead of about 1.5%. This configuration maintains a probability of convergence of over 85% when 14 faults are injected on average at random points during the solve phase. It takes on average 37 faults per solve before the probability of converging of the low cost configuration to drop below 75%.

2.4. Error Propagation. SDC detection relies on detecting deviations from the normal correct state of data or application execution. Understanding how the error due to a fault propagates to corrupted enough of the application's state to be detected can allow for better design and placement of SDC detectors. Error propagation is often considered in the context of mathematical interaction/operations on floating-point data; however, floating-point operations are only a fraction of operations in a typical HPC application. Propagation from these non floating-point operations is key to fully understand how SDC propagates inside HPC applications. To track SDC propagation in detail, I built a LLVM compiler pass that simulates lock-step execution between a non-faulty gold version of the application and an error-prone faulty version. Lock step execution allows for detailed comparisons of intermediate results at a load/store level granularity. This micro-level view of SDC propagation allows for detection and understanding of latencies associated with detection, detection of diverges in control flow, and deviations in partial results. Micro-level propagation results are useful for resiliency schemes that focus on instruction duplication or alteration. Micro-level propagation is complimented by tracking propagation at the application level variable level through macro-level propagation. Macro-level propagation looks at deviation in application state variables. It allows for statistics to be calculated on the deviation of each state variable which can be used to select which state variables are candidates for replication or require SDC checks. An early version of this work was presented as a poster at CLUSTER 2015 [3]. A paper is being drafted on a more detailed version in which we explore how different classifications of instructions such as floating-point and fixed-point effect error propagation. In addition, we explore how compiler optimizations impact error propagation.

2.5. Lossy compression. Data movement is becoming a bottleneck for HPC applications and can occur at all levels in the memory hierarchy. The parallel file system is a primary data movement bottleneck, but is required for persistence of datasets for visualization and checkpoint restart. Reducing the volume of data sent to the parallel file system via compression techniques can mitigate the effects of this bottleneck. Traditional lossless compression schemes are often ill suited for HPC floating-point datasets [9]. Floating-point specific compressors do improve compression factors, but significantly higher compression factors can be achieved with lossy compression. Lossy compression achieves high compression factors by adding small controllable amounts of error into the data. The larger the error tolerance, the higher the compression factor. Key to understanding how lossy compression can be used is a firm understanding of the compression tolerance. Prior work that investigates lossy compression for checkpoint restart [6][5][7] relies on either trial and error based approaches guided by a detailed understanding of what is being modeled and investigating where this breaks down.

Many HPC applications approximate the solution to PDEs or ODEs. These approximations are accurate up to truncation error of the numerical method used in the problem parameterized by the problem's spatial and temporal discretization. My work interprets error added via lossy compression as numerical error and seeks to mask lossy compression error in high-order truncation error that is not considered significant enough to impact the simulation. For a given numerical simulation, if the truncation error is α then we select a compression tolerance ϵ such that $\epsilon < \alpha$. This ensures that the result of the simulation restarted from a lossy checkpoint is numerically equivalent to a simulation restarted from an uncompressed checkpoint. Results show that selecting compression tolerances in this way for the production level codes PlasComCM and Nek5000, allows error in the state variables to be bounded less than the simulation's truncation

error for current system mean time between failure (MTBF). This holds for multiple restarts and for system MTBF of a few minutes representing a worst case exascale machine.

3. Future Work. Going forward, I plan to focus on further exploration of how lossy compression can be used. I plan on extending my work on error tolerance selection with other methodologies for selecting error tolerances that leverage other quantities of interest that are important to the simulation such as relative error and statistical properties of the data. Furthermore, understanding how selection of boundary conditions and domain geometry influence the simulation's ability to retain or remove lossy compression error influences how often restarting from lossy compressed checkpoints are possible.

Understanding how simulations behave with error in them can lead to further optimizations when selecting lossy compression tolerances. If it is known that the simulation remove error then larger compression tolerances can be used improving how well the data can be compressed. Understanding the best practices of how to use lossy compression and where to use it are open questions, and my goal during the remainder of my thesis is to lay the foundation such that lossy compression can be seen as an avenue for program optimization rather than just a research project.

REFERENCES

[1] J. Calhoun, L. Olson, and M. Snir, *FlipIt: An LLVM based fault injector for HPC*, in Proceedings of the 20th International Euro-Par Conference on Parallel Processing (Euro-Par '14), 2014.

[2] J. Calhoun, L. Olson, M. Snir, and W. D. Gropp, *Towards a more fault resilient multigrid solver*, in Proceedings of the Symposium on High Performance Computing, HPC '15, San Diego, CA, USA, 2015, Society for Computer Simulation International, pp. 1–8, http://dl.acm.org/citation.cfm?id=2872599.2872600.

[3] J. Calhoun, M. Snir, L. Olson, and M. Garzaran, *Understanding the propagation of error due to a silent data corruption in a sparse matrix vector multiply*, in Proceedings of the 2015 IEEE International Conference on Cluster Computing, CLUSTER '15, Washington, DC, USA, 2015, IEEE Computer Society, pp. 541–542, doi:10.1109/CLUSTER.2015.101, http://dx.doi.org/10.1109/CLUSTER.2015.101.

[4] M. Casas, B. R. de Supinski, G. Bronevetsky, and M. Schulz, *Fault resilience of the algebraic multi-grid solver*, in Proceedings of the 26th ACM international conference on Supercomputing, ICS '12, New York, NY, USA, 2012, ACM, pp. 91–100, doi:10.1145/2304576.2304590, http://doi.acm.org/10.1145/2304576.2304590.

[5] D. Laney, S. Langer, C. Weber, P. Lindstrom, and A. Wegener, *Assessing the effects of data compression in simulations using physically motivated metrics*, in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '13, New York, NY, USA, 2013, ACM, pp. 76:1–76:12, doi:10.1145/2503210.2503283, http://doi.acm.org/10.1145/2503210.2503283.

[6] X. Ni, T. Islam, K. Mohror, A. Moody, and L. V. Kale, *Lossy compression for checkpointing: Fallible or feasible?*, in Poster Session of the 2014 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '14, Washington, DC, USA, 2014, IEEE Computer Society.

[7] N. Sasaki, K. Sato, T. Endo, and S. Matsuoka, *Exploration of lossy compression for application-level checkpoint/restart*, in Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS '15, Washington, DC, USA, 2015, IEEE Computer Society, pp. 914–922, doi:10.1109/IPDPS.2015.67, http://dx.doi.org/10.1109/IPDPS.2015.67.

[8] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. A. Chien, P. Coteus, N. A. DeBardeleben, P. C. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. Munson, R. Schreiber, J. Stearley, and E. V. Hensbergen, *Addressing failures in exascale computing*, International Journal of High Performance Computing Applications, 28 (2014), pp. 127 – 171, doi:10.1177/1094342014522573.

[9] S. W. Son, Z. Chen, W. Hendrix, A. Agrawal, W. keng Liao, and A. Choudhary, *Data compression for the exascale computing era - survey*, Supercomputing frontiers and innovations, 1 (2014), http://superfri.org/superfri/article/view/13.