# Understanding Ineffectiveness of the Application-Level Fault Injection

Luanzheng Guo
U. of California, Merced
lguo4@ucmerced.edu

Jing Liang
U. of California, Merced
jliang27@ucmerced.edu

Dong Li
U. of California, Merced
dli35@ucmerced.edu

## ABSTRACT

Extreme-scale applications are at a significant risk of being hit by soft errors on supercomputers, as the scale of these systems and the component density continues to increase. In order to better understand soft error vulnerabilities in those applications, the application-level fault injection is widely employed to evaluate applications. This poster reveals that the application-level fault injection has some inherent uncertainties due to the random nature of fault injection. First, the fault injection result has a strong correlation with the number of fault injection tests. What is a good number of fault injection tests is uncertain. Second, given a specific application, the fault injection result can vary as the input problem of the application varies. How to interpret the fault injection result is uncertain. Those uncertainties can make fault injection ineffective for accurately modeling application vulnerability.

## 1. INTRODUCTION

Resilience is one of the major design goals for extreme-scale systems. Looking forward to Exascale, the sheer scale of components and the move toward heterogeneous architectures and shrinking feature size of hardware will compound the resilience challenge. One especially difficult problem in resilience is soft errors: a one-time, unpredictable event that results in bit flips in memory and errors in logic circuit outputs that corrupt and contaminate a computing system's state. A soft error can alter the results of applications silently without any omen; a soft error can cause application crashes and extend application execution time. Because of the highly negative impact of soft errors on applications, we must better understand the impact of soft errors on large-scale applications.

The application-level fault injection is one of the most common methods to study the application resilience to soft errors. With the application-level fault injection, artificial faults are randomly injected into application variables or computation logic, and the application vulnerability is then evaluated based on the application responses [2, 3, 1, 4]. This methodology usually performs a large amount of fault injection tests. Among $N$ fault injection tests, if $M$ of them have correct application execution and outcome, then the application vulnerability (i.e., the fault injection result) is calculated as $M/N$. A lower value of the fault injection result indicates that the application is more vulnerable. We have seen LLVM-based fault injection tools [2, 3, 1] and binary instrumentation-based fault injection tools [4, 6]. Many applications and algorithms have been evaluated based on the application-level fault injection, such as AMG [3], a couple of iterative methods [5], and Nek5K [4].

The application-level fault injection has some inherent uncertainties due to the random nature of fault injection. Given an application, how many fault injection tests should we do to make a statistically significant conclusion on the application vulnerability? Also, given an application, the current fault injection practices often study its vulnerability with a single input problem. Does the application vulnerability vary with the input problem? Those uncertainties can make fault injection ineffective for accurately modeling application vulnerability.

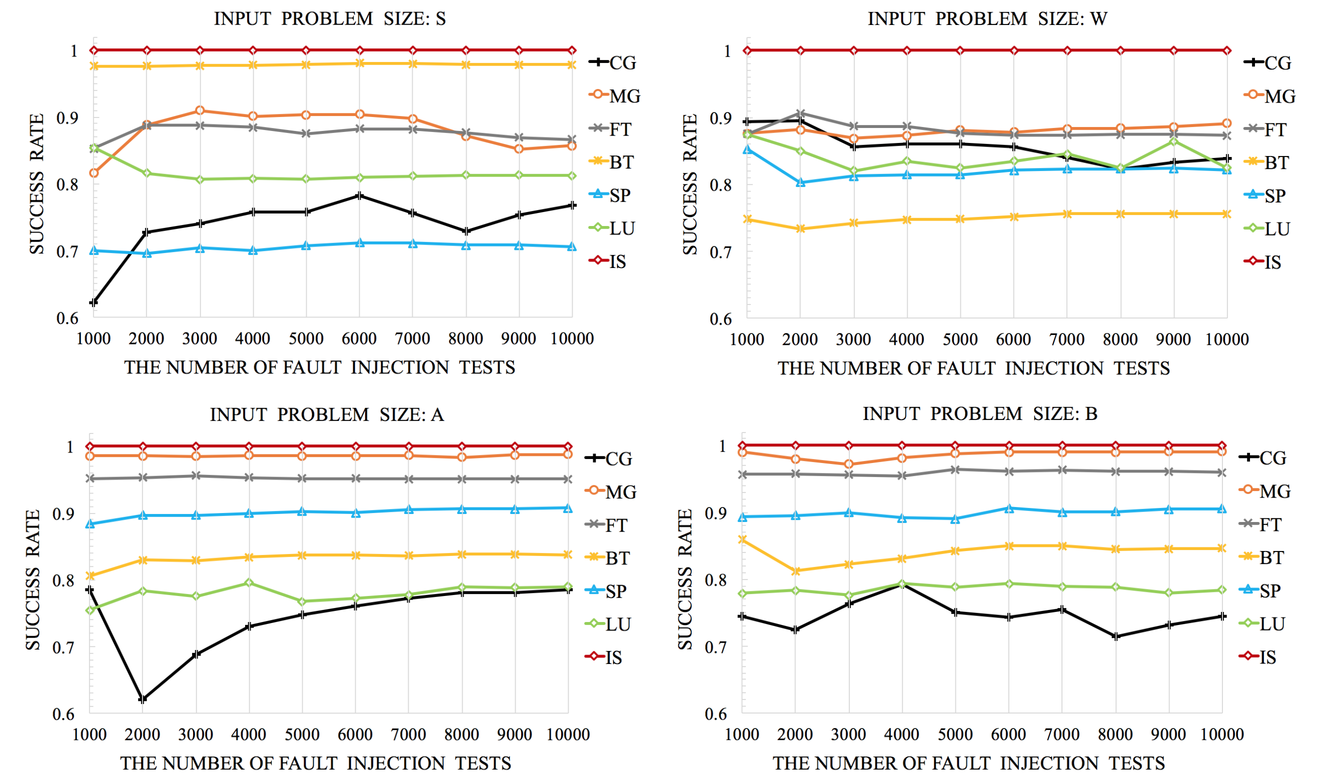
In this poster, we explore the answers to the above questions and aim to understand the inherent ineffectiveness of application-level fault injection. We use an LLVM-based fault injection tool, and extensively perform fault injection on a series of benchmarks with different numbers of fault injection tests and different input problems. We have two interesting conclusions based on our tests. First, the fault injection results have a strong correlation with the number of fault injection tests: the fault injection results of some applications are highly sensitive to the umber of fault injection tests, while the fault injection results of some applications are relatively independent of the number of fault injection tests. Second, given a specific application, the fault injection result can vary as the input problem of the application varies. Our conclusions provide important guidance on the future application-level fault injection practice.

## 2. METHODOLOGY

We employ a LLVM-based fault injection tool, FlipIt [2], to study seven NAS parallel benchmarks (CG, MG, FT, BT, SP, LU, IS). With FlipIt, for each fault injection test, we randomly select an instruction and then randomly flip a bit in the output operand. We always use single-bit flip for our fault injection tests. We inject faults into the critical functions of the benchmarks (particularly, conj_grad for CG, mg3P for MG, fftXYZ for FT, x_solve for BT, x_solve for SP, ssor for LU, and randlc for IS). For each benchmark, we study five different input problems (S, W, A, B, and C). For each input problem, we do ten sets of fault injection tests. The number of fault injection tests in the ten sets ranges from 1000 to 10,000 with a stride of 1000.

## 3. EXPERIMENT RESULTS

Figures 1 show the fault injection results (shown as "success rate" in the $y$ axis). We first study the sensitivity of the fault injection results to the number of fault injection tests. We find that the fault injection result of CG is highly sensitive to the number of fault injection tests. For the input problem A, the fault injection results are 62% for 2000 fault inject tests and 75% for 10,000 fault injection tests. There is 13% difference. In addition, when the number of fault injection tests is 1000 (using input problem A), we find that CG demonstrates higher resilient than MG (0.78 for CG vs. 0.76 for MG). However, when the number of fault injection



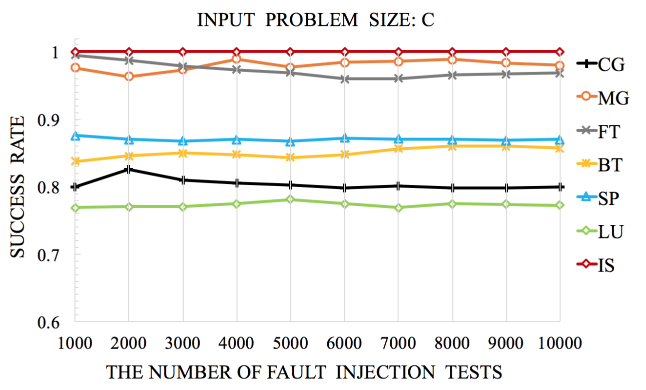


**Figure 1: The fault injection results for seven NPB benchmarks.**

tests is 2000, we make a totally opposite conclusion: MG is more resilient than CG (0.62 for CG vs. 0.78 for MG). This observation is a clear demonstration of the ineffectiveness of using fault injection to study the application vulnerability. Besides CG, the fault injection result of MG also has some obvious variance when the input problem size is S. For other benchmarks (IS, FT, BT, SP, LU), their fault injection results are relatively stable (less than 10% variance) across different numbers of fault injection tests. This suggests that 1000 fault injection test (the least number of fault injection tests in our experiments) is good enough to study their vulnerability.

We further study the impact of input problem size on the application vulnerability. We focus on the results of 10,000 fault injection tests. For input problem W, the fault injection result of BT is 75%, while for other input problems, the fault inject results are all about 85%. SP also demonstrates high sensitivity: the fault injection result varies from 70% to 90% across input problem sizes. For CG, FT, MG, LU, and IS, the fault injection results are relatively stable across different input problem sizes.

**Conclusions:** We aim to understand the inherent ineffectiveness of application-level fault injection. We hope that this poster can help us improve the common practice of fault injection and enable a more accurate understanding of application vulnerability.

## 4. REFERENCES

[1] R. A. Ashraf, R. Gioiosa, G. Kestor, R. F. DeMara, C.-Y. Cher, and P. Bose. Understanding the Propagation of Transient Errors in HPC Applications. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015.

[2] J. Calhoun, L. Olson, and M. Snir. Flipit: An LLVM based fault injector for HPC. In *Euro-Par Parallel Processing Workshops*, 2014.

[3] M. Casas, B. R. de Supinski, G. Bronevetsky, and M. Schulz. Fault Resilience of the Algebraic Multi-grid Solver. In *International Conf. on Supercomputing*, 2012.

[4] D. Li, J. S. Vetter, and W. Yu. Classifying Soft Error Vulnerabilities in Extreme-Scale Scientific Applications Using a Binary Instrumentation Tool. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2012.

[5] M. Shantharam, S. Srinivasmurthy, and P. Raghavan. Characterizing the Impact of Soft Errors on Iterative Methods in Scientific Computing. In *International Conference on Supercomputing (ICS)*, 2011.

[6] J. Wei, A. Thomas, G. Li, and K. Pattabiraman. Quantifying the Accuracy of High-Level Fault Injection Techniques for Hardware Faults. In *International Conference on Dependable Systems and Networks*, 2014.