POSTER: Hardening Selective Protection across Multiple Program Inputs for HPC Applications

Yafan Huang
University of Iowa
Iowa City, IA, USA
yafan-huang@uiowa.edu

Shengjian Guo
Baidu Security
Sunnyvale, CA, USA
sjguo@baidu.com

Sheng Di
Argonne National Laboratory
Lemont, IL, USA
sdi1@anl.gov

Guanpeng Li*
University of Iowa
Iowa City, IA, USA
guanpeng-li@uiowa.edu

Franck Cappello
Argonne National Laboratory
Lemont, IL, USA
cappello@mcs.anl.gov

Abstract

With the ever-shrinking size of transistors and increasing scale of applications, silent data corruptions (SDCs) have become a common yet serious issue in HPC applications. Selective instruction duplication (SID) is a popular fault-tolerance technique that can obtain a high SDC coverage with low-performance overhead, as it selects the most vulnerable parts of a program for protection with priority. However, existing studies of SID are confined to single program input in the evaluation, assuming that the error resilience of the program remains similar across inputs, leading to a drastic loss of SDC coverage from SID when the protected program runs different inputs. Hence, we proposed Sentinel, an automated compiler-based framework to mitigate the loss of SDC coverage. Evaluation results show that Sentinel can effectively mitigate the loss of SDC coverage (up to 97.00%) across multiple inputs, which significantly hardens existing SID techniques.

CCS Concepts: • **Software and its engineering → Compilers**; • **Computer systems organization → Reliability**.

Keywords: Error Resilience, Fault Injection, Compiler, High Performance Computing

1 Introduction

Silent data corruption (SDC) is recognized as one of the most severe types of soft error as it propagates in program execution and finally corrupts program output without noticeable symptoms. Researchers have observed that only a portion of program instructions are responsible for almost all the SDCs in a program. Thus, protecting the most vulnerable parts of the program may be sufficient to achieve high SDC coverage with relatively low overhead. In light of this, *selective instruction duplication* (SID) has been proposed to mitigate SDCs in programs [1, 2]. SID chooses the most vulnerable instructions in a program and duplicates them against SDCs in the program execution.

However, existing studies in SID often confine themselves to one program input in the protection evaluation, assuming that the error resiliency of the program remains similar under different inputs. Nevertheless, we observe that such assumption cannot always hold, leading to a significant loss in SDC coverage when a protected program runs with various inputs, which seriously compromises HPC reliability in a production environment. Industry experiences with a similar view have recently been released, disclosing much higher SDC rates in their large-scale application executions, even

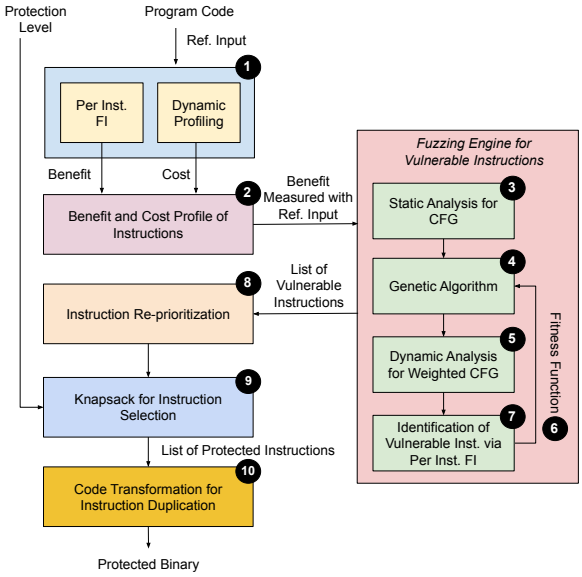
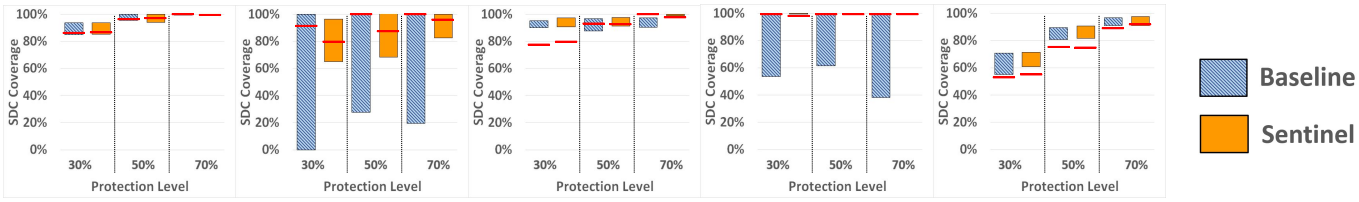


Figure 1. Workflow of Sentinel

*Corresponding author.

PPoPP '22, April 2–6, 2022, Seoul, Republic of Korea

ACM ISBN 978-1-4503-9204-4/22/04.
https://doi.org/10.1145/3503221.3508414



(a) Pathfinder **(b)** Needle **(c)** Particlefilter **(d)** kNN **(e)** HPCCG

Figure 2. Mitigation of losing SDC coverage by Sentinel, compared with the existing SID method (Red bars indicate expected SDC coverage provided by each technique)

the programs are protected. To tackle this issue, we propose Sentinel, an compiler-based technique that improves the overall SDC coverage for SID through static analysis and dynamic input fuzzing. The experiment results demonstrate our proposed Sentinel can effectively mitigate the loss of SDC coverage across multiple program inputs.

2 Design of Sentinel

Figure 1 shows the overall workflow of Sentinel. The core target of Sentinel is to efficiently identify the vulnerable instructions across program inputs. Besides SID preparation steps (❶ and ❷), we leverage the static analysis in Sentinel to generate the control-flow graph (CFG) for a given program (❸). With a genetic algorithm search engine (❹) and dynamic profiling (❺) of the program execution, Sentinel computes a weighted CFG list for input. The weighted CFG list represents the unique execution paths under the input and is used by the fuzzing engine to differentiate the program execution from all other inputs seen in the past search. In this way, Sentinel can identify those vulnerable instructions with fewer inputs, hence requiring much less per-instruction FIs (❼) to be performed. Then Sentinel re-prioritizes those instructions by updating their benefits with the upper bounds measured among the generated inputs (❽), to let SID prioritize them in the instruction selection phase (❾). Finally, Sentinel completes instruction duplication and generates the protected binary of the program.

3 Evaluation

Figure 2 demonstrates the effectiveness of Sentinel in mitigating the loss of SDC coverage, by comparing Sentinel with the baseline (state-of-the-art SID method). As shown in the figure, the ranges of SDC coverage provided by Sentinel are much shorter in almost all the benchmarks at every protection level, compared with the baseline. The lower bounds of the coverage are much higher than those in the baseline. Overall, averaging over all the benchmarks at the 3 protection levels, Sentinel mitigates 97% of the loss of SDC coverage in the existing SID. Another important observation is that, unlike the baseline, the minimum coverage provided by Sentinel always increase as the protection level increases in each benchmark. For example, in kNN, the minimum coverage provided by Sentinel is 99.54% at 30% protection level, 100.00% at 50% and 70.00% protection level. In contrast, the coverage ability of the baseline may decrease unexpectedly with increasing protection levels. That is, Sentinel provides more predictable SDC coverage as the protection level elevates, which is critical for developers to improve the software resilience and meet the reliability target. Table 1 examines the percentage of random inputs that still have SDC-coverage-loss issue when using Sentinel. By checking the tables, we observe that the percentage of random inputs that lead to losing coverage turn much lower in Sentinel than that in the baseline. This shows that in production environment, by applying Sentinel, the protected applications will less likely experience the loss of SDC coverage. For the time taken to run Sentinel, it takes on average 8.04 hours to complete the entire process for each benchmark.

Table 1. Percentage of Random Inputs that Result in the Loss of SDC Coverage in Sentinel

Benchmark	30% Level	50% Level	70% Level
Pathfinder	13.33%	3.33%	3.33%
Needle	3.33%	3.33%	3.33%
Particlefilter	0.00%	56.67%	0.00%
kNN	0.00%	0.00%	0.00%
HPCCG	0.00%	0.00%	3.33%
Average	3.33%	12.67%	2.00%
Average (Baseline)	23.33%	42.67%	44.67%

4 Acknowledgment

The material was supported by the U.S. Department of Energy, Office of Science under contract DE-AC02-06CH11357.

References

[1] Chun-Kai Chang, Guanpeng Li, and Mattan Erez. 2019. Evaluating compiler ir-level selective instruction duplication with realistic hardware errors. In *2019 IEEE/ACM 9th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS)*. IEEE, 41–49.

[2] Qining Lu, Karthik Pattabiraman, Meeta S Gupta, and Jude A Rivers. 2014. SDCTune: a model for predicting the SDC proneness of an application for configurable protection. In *Proceedings of the 2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. 1–10.