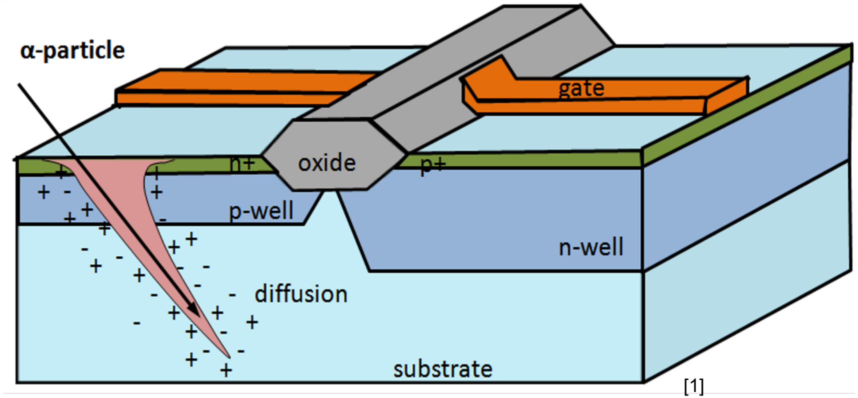
SC22 Dallas, TX | hpc accelerates.

Mitigating Silent Data Corruption in HPC Applications across Multiple Program Inputs

Yafan Huang, Shengjian Guo, Sheng Di, Guanpeng Li, Franck Cappello

The University of Iowa · Argonne National Laboratory · Baidu USA

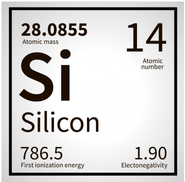
Soft Error



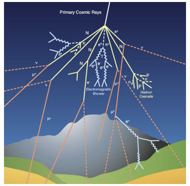
Logic Values in Hardware

Before	0	1	0	1	1
After	0	1	**1**	1	1

bit flip



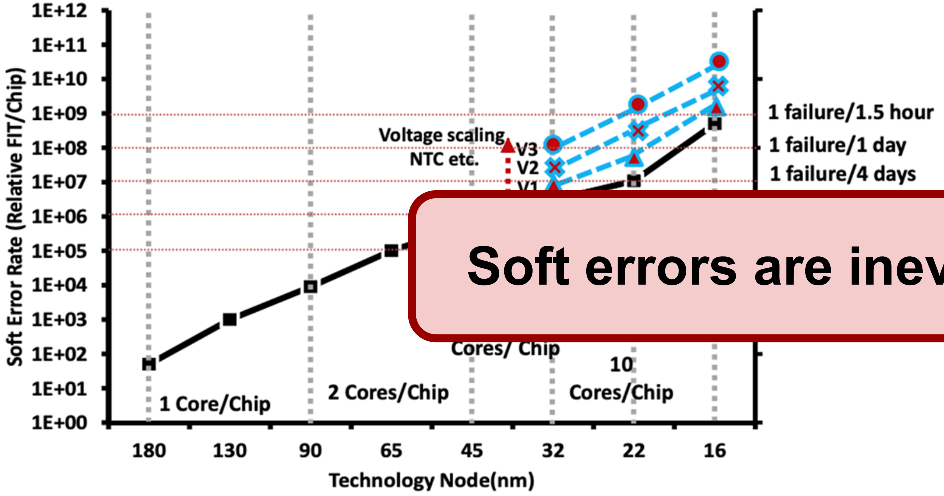
Silicon Decay



Cosmic Radiation

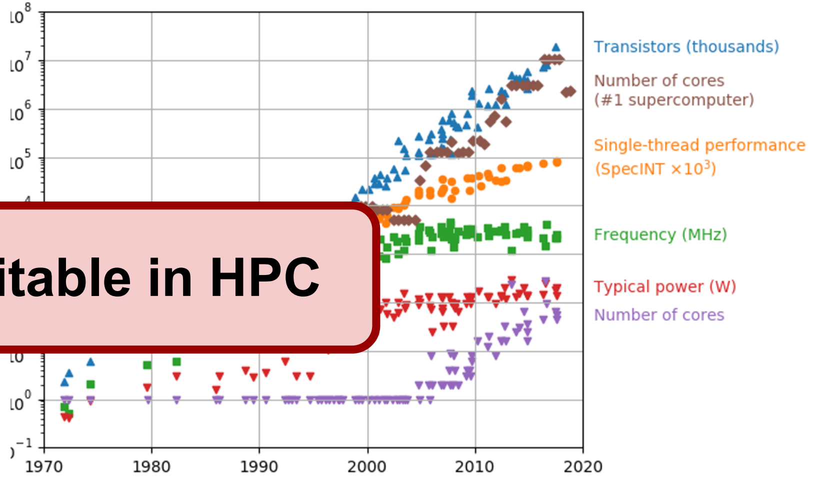
[1] https://labs.engineering.asu.edu/mps-lab/research/error-resilience/

Soft Errors in HPC



Shrinking hardware technology[1]

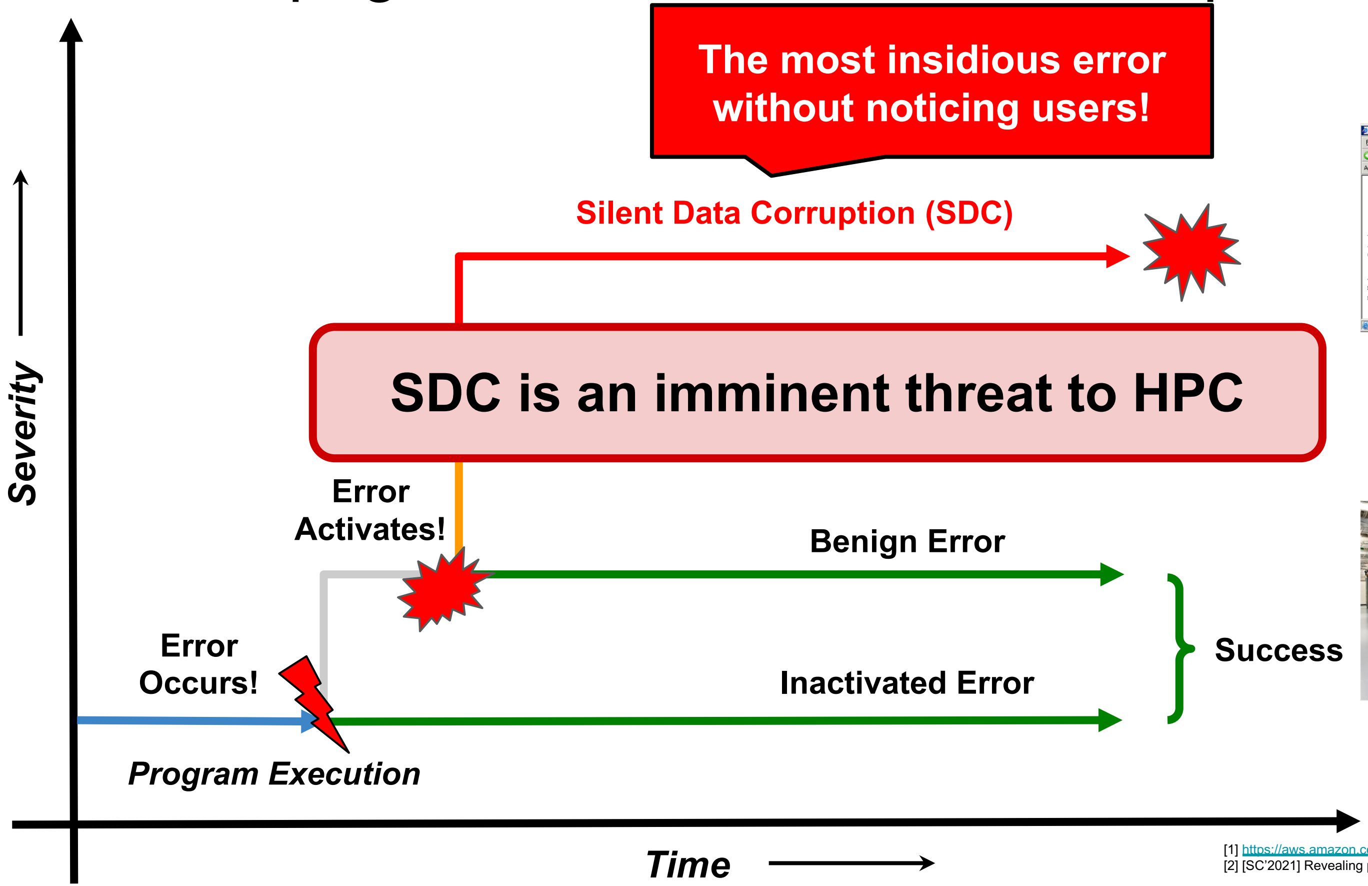
Soft errors are inevitable in HPC

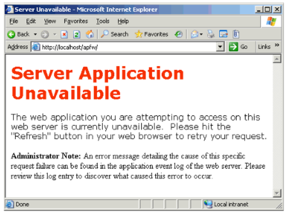


Increasing HPC system scales[2]

[1] [ToC'2016] A Case for Acoustic Wave Detectors for Soft-Errors
[2] https://github.com/karlrupp/microprocessor-trend-data
[3] [DSN'2014] Lessons Learned from the Analysis of System Failures at Petascale: The Case of Blue Waters

Error Propagation and Silent Data Corruption (SDC)





Amazon S3 Incident[1]

Summit Reliability Report[2]

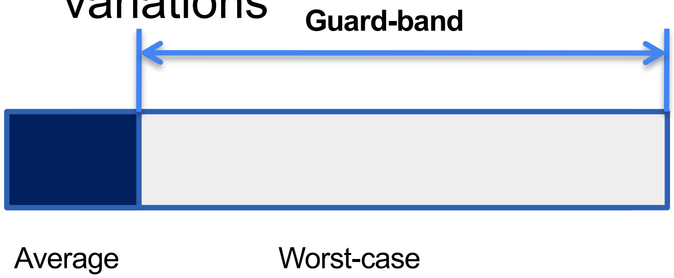
[1] https://aws.amazon.com/message/41926/
[2] [SC'2021] Revealing power, energy and thermal dynamics of a 200PF pre-exascale supercomputer

Traditional Solutions

- In memory: Error Correction Code (ECC)
- In pipeline: Hardware means

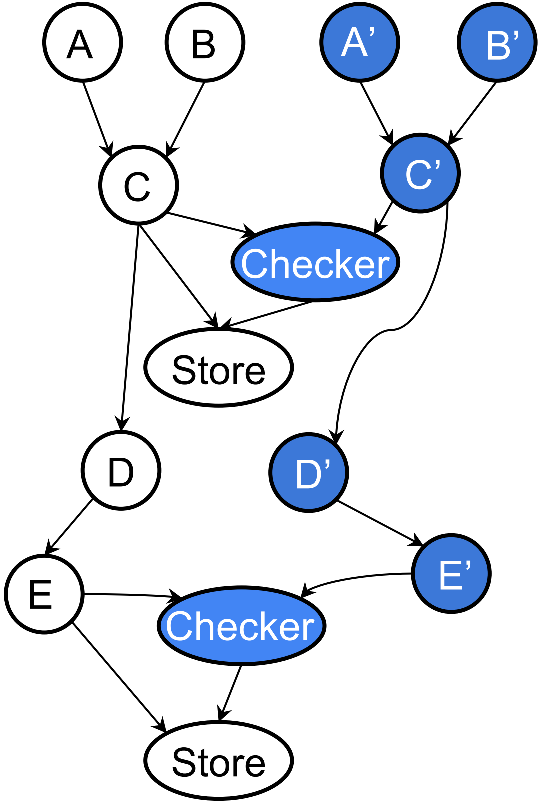
Very expensive to deploy in practice.

- **Circ**
  Waste
  between average and
  worst-case widens due to
  variations



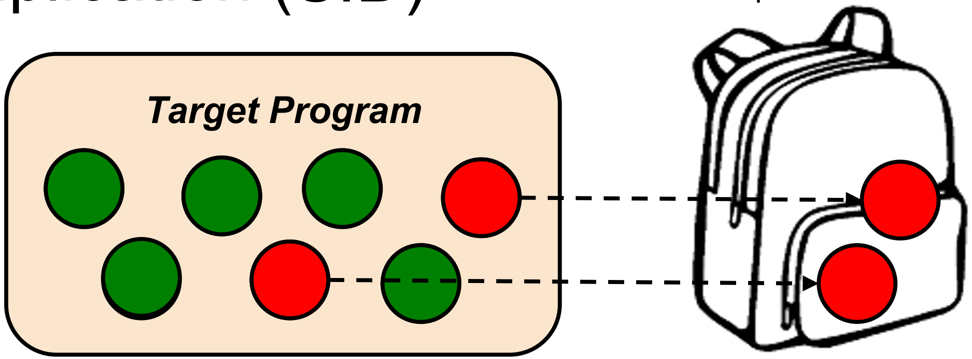
- **ation**
  (i.e.,
  DMR) can result in 2X
  slowdown and/or energy
  consumption

Selective Instruction Duplication (SID)

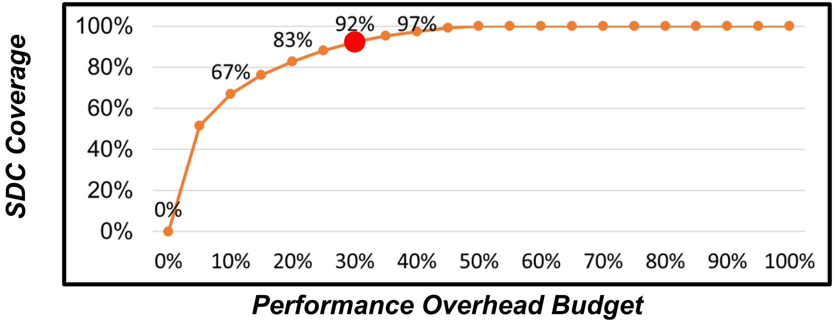


Instruction Sequence

Knapsack Problem



Target of SID: Obtaining maximum SDC coverage under given performance overhead budget.



The cost-benefit curve of SID (Needle)

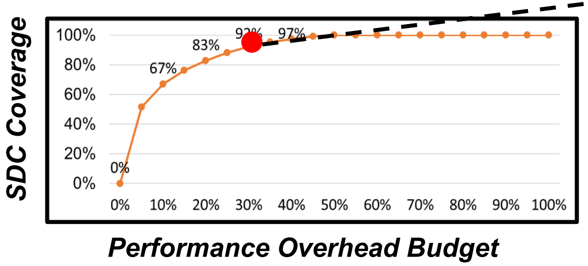
The Problem: Input Variation

Single Input

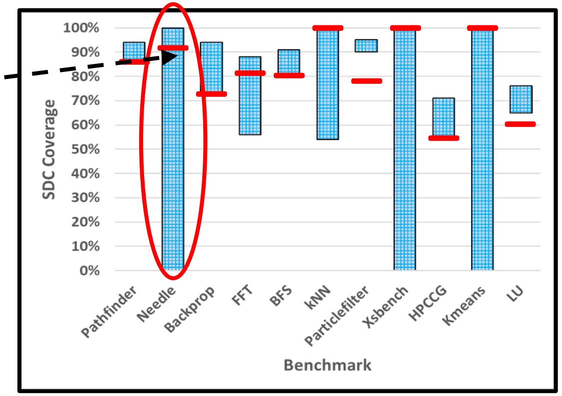
Existing SID

Assumption

Error propagation behaviors in a program across the different inputs remain rather similar.



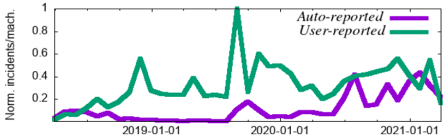
The cost-benefit curve of SID (Needle)



SDC Coverage Variation of 50 inputs under 30% Performance Overhead Budget

- SDC Coverage varies from **0%** to **100%**.
- Expected SDC Coverage is way too optimistic.
- **37.58%** inputs lead to loss of SDC coverage.

Google



9x user-reported SDC[1]

[1] [HotOS'2021] Cores that don't count.

Root Causes

Incubative Instruction

Instructions that experience significant variance in SDC probabilities across different program inputs.

Input A

```
...
%11 = load i64* %out
%12 = icmp slt i64 %11, 50
br label %13
...
```

Instruction Sequen

SDC Prob. = 0%
No need protect.

Input B

```
...
%11 = load i64* %out
%12 = icmp slt i64 %11, 50
br label %13
...
```

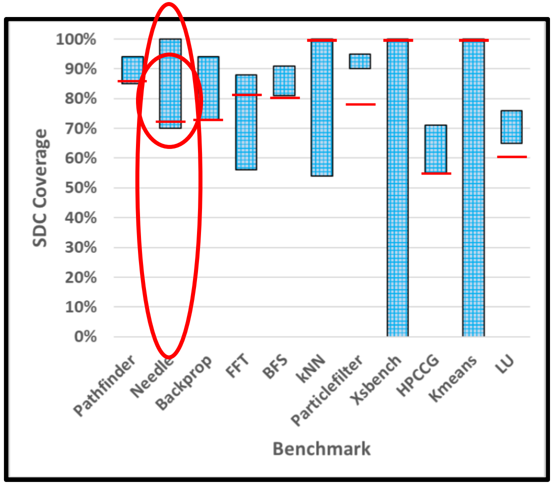
SDC Prob. = 17%
Need protect.

- No SDCs under **test input**, but SDC happens under **a different input**.
- Input variation changes the program execution behaviors (e.g. control-flow), hence changes error propagation behaviors of different instructions.

Goal and Insights

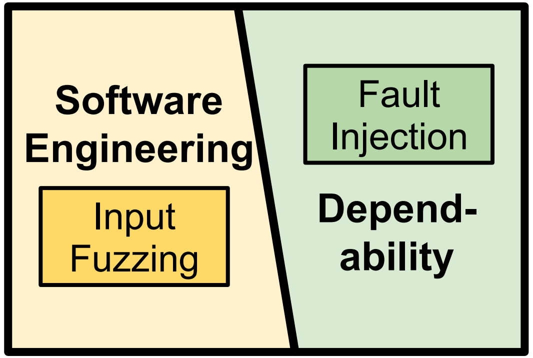
Our Goal:

- **Minimize SDC coverage variation.**
- **Make expected SDC coverage closer to the most conservative one.**

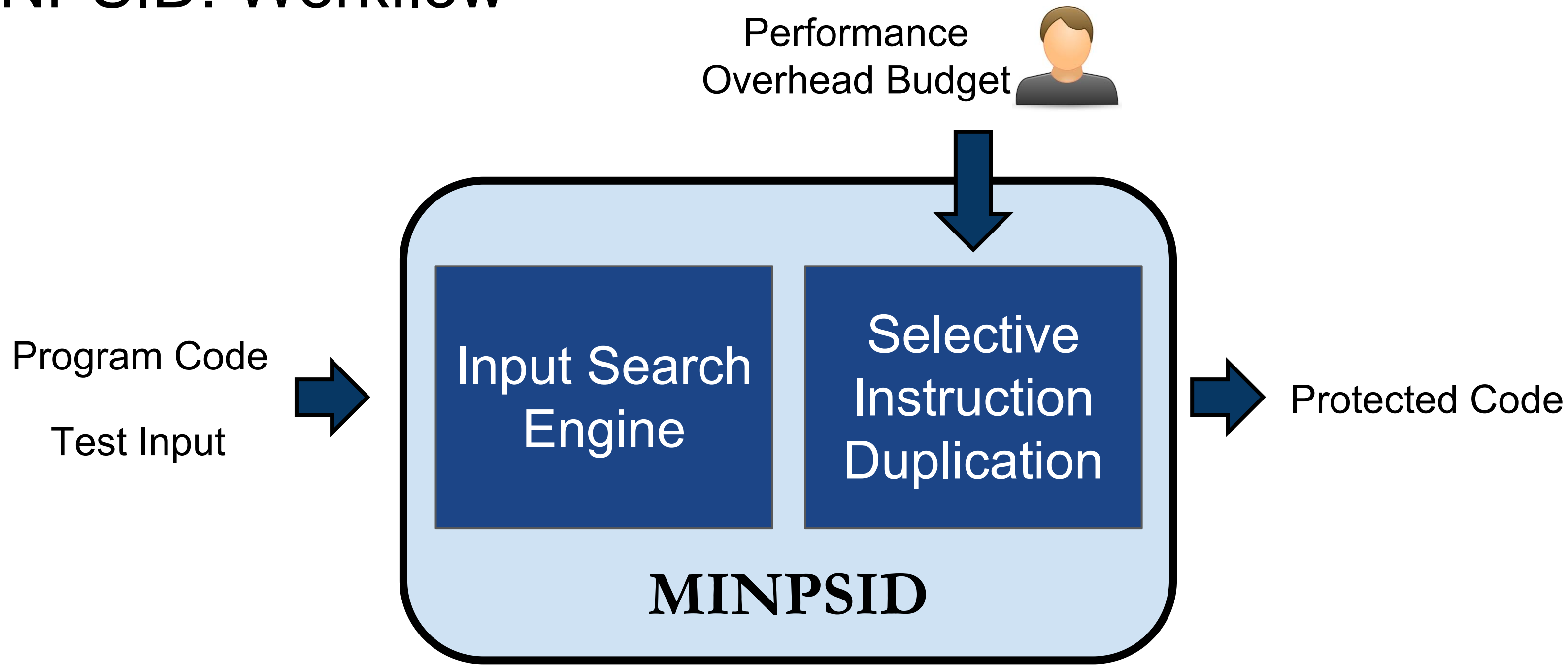


Key Insights

Identify program inputs that maximize the control-flow variances.

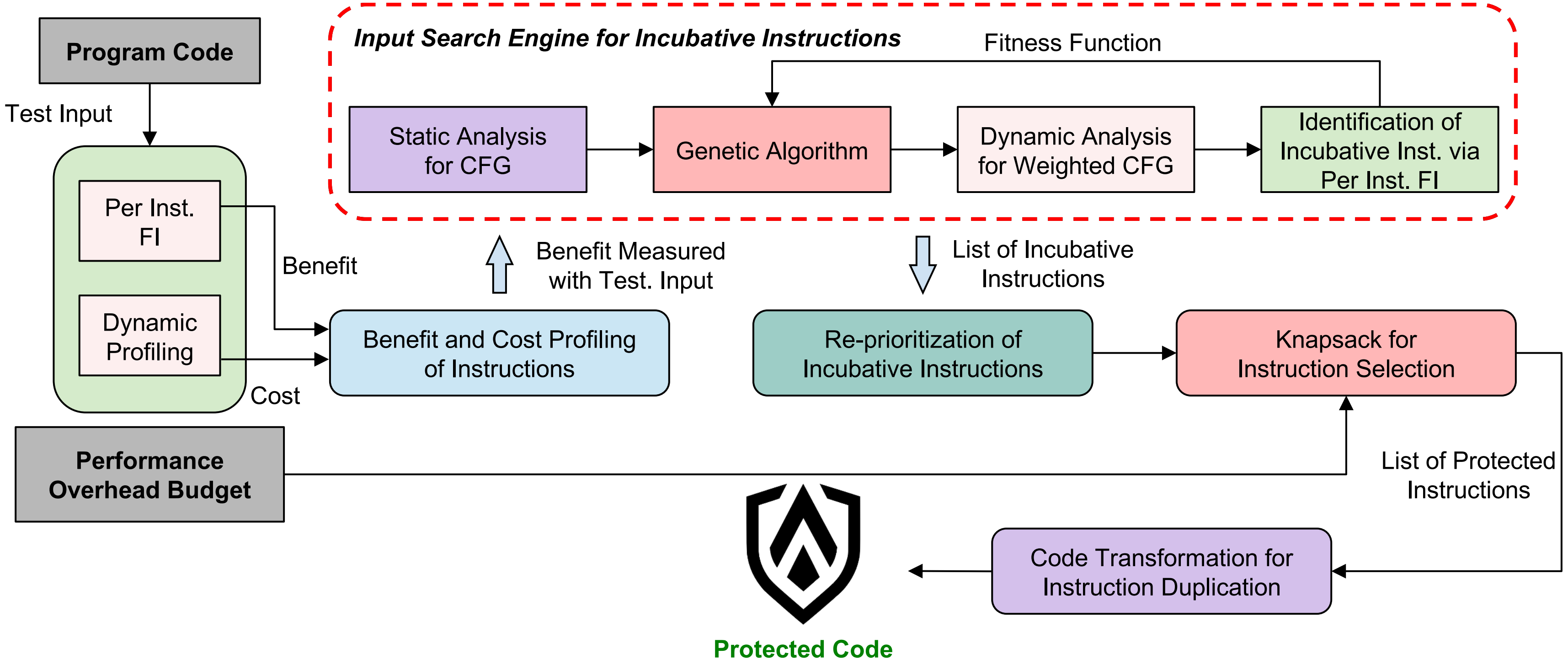


MINPSID: Workflow



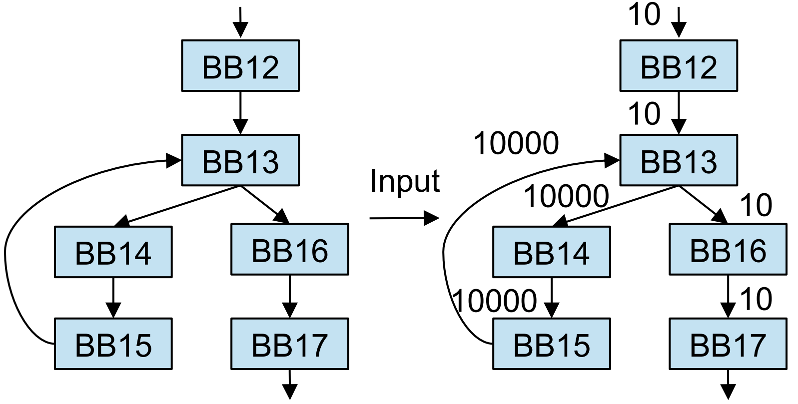
(**M**ulti-**In**put-Hardened **S**elective **I**nstruction **D**uplication)

MINPSID: Our Approach



MINPSID: Input Search Engine

The search of genetic algorithm is drove by the fitness function.



Static CFG

Weighted CFG

Index	Value
...	
BB12	10
BB13	10000
BB14	10000
BB15	10000
BB16	10
BB17	10
...	

Indexed CFG List

Fitness Function:

$$S_L = \frac{1}{|M|+1} \sum_{j=0}^{M} \sqrt{\sum_{n=1}^{N} |i_n - b_{jn}|^2}$$

S_L: Fitness score
M: Number of historical inputs
N: Number of inst. in CFG
$|i_n - b_{jn}|$: Euclidean distance between two inst.

Quantify a program execution with an input.

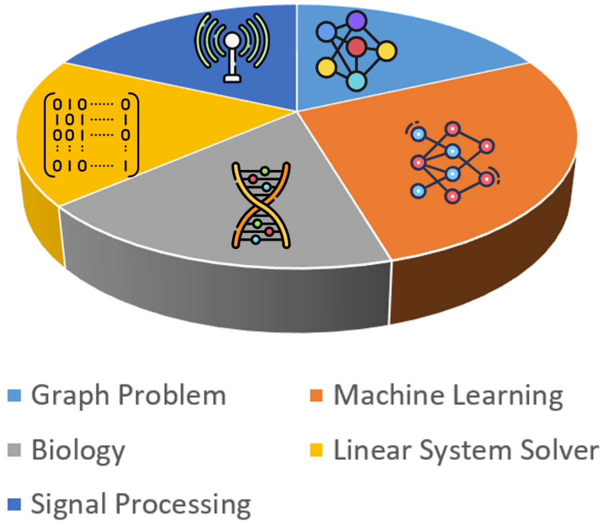
- **Weighted CFG:** Generate an indexed CFG list of a program input.

Guide GA search

- **Fitness Function:** Calculate average Euclidean distance between current input with all historical searched inputs.

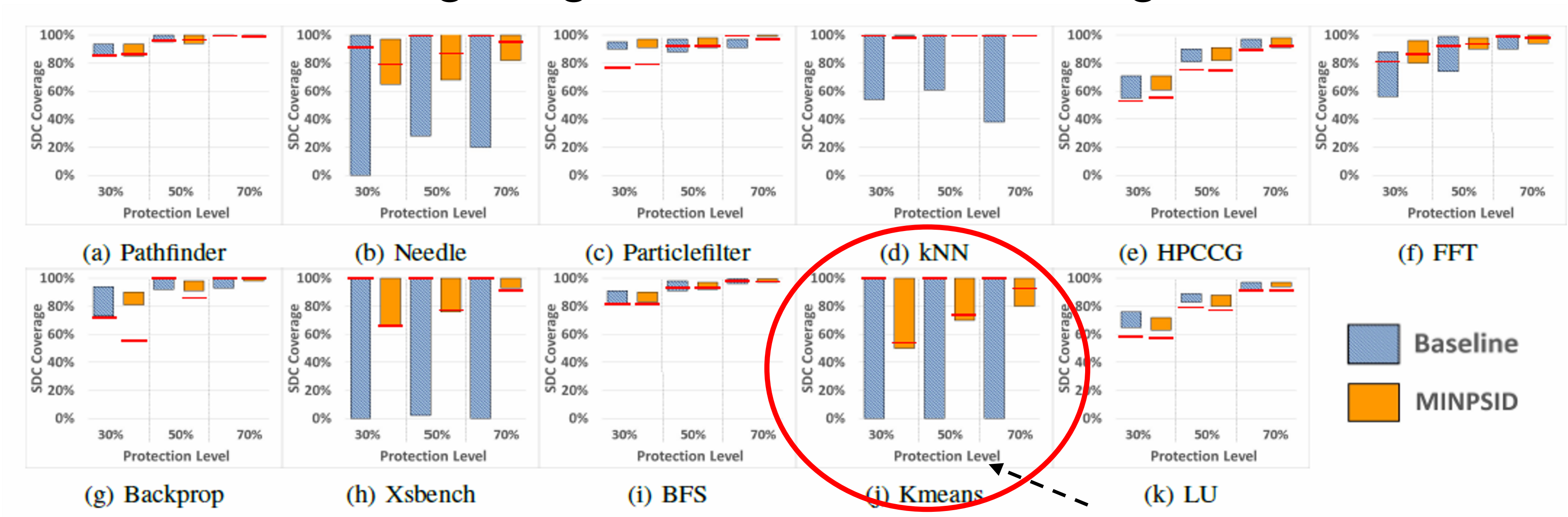
Evaluation: Experimental Setup

- Benchmark
  - 11 open-source benchmarks
- Baseline Technique
  - Selective instruction duplication[1]
- Fault Model
  - Single bit-flip injections - accurate[2]
  - Errors in computation units/data path
  - One fault per program execution
  - User LLFI[3] for fault injection
- Input Generation
  - Random inputs
    - 50 inputs for each benchmark
  - Real-world inputs
    - KONECT Graph Collection
    - Kaggle Competition Dataset



[1] [CASES'2014] SDCTune: A model for predicting the SDC proneness of an application for configurable protection
[2] [DSN'2017] One Bit is (Not) Enough: An Empirical Study of the Impact of Single and Multiple Bit-Flip Errors
[3] [QRS'2015] LLFI: An Intermediate Code-Level Fault Injection Tool for Hardware Faults

Evaluation: Mitigating Loss of SDC Coverage

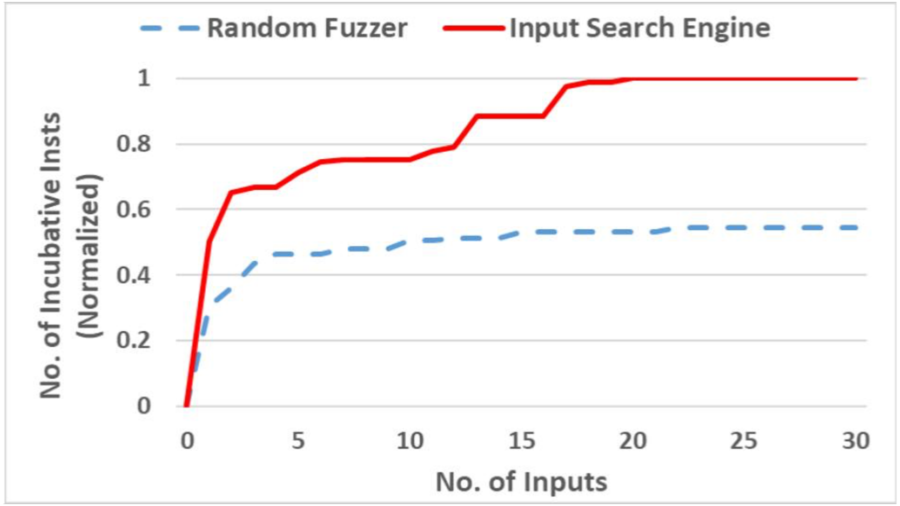


"Protection level" here means "Performance Overhead Budget"

- The SDC coverage variation across different inputs is significantly (**74.23%**) reduced.
- The expected SDC coverage is closer to the most conservative one, reducing **97%** loss of SDC coverage.
- Only **8.36%** inputs lead to the loss of SDC coverage (**37.58%** for baseline SID).

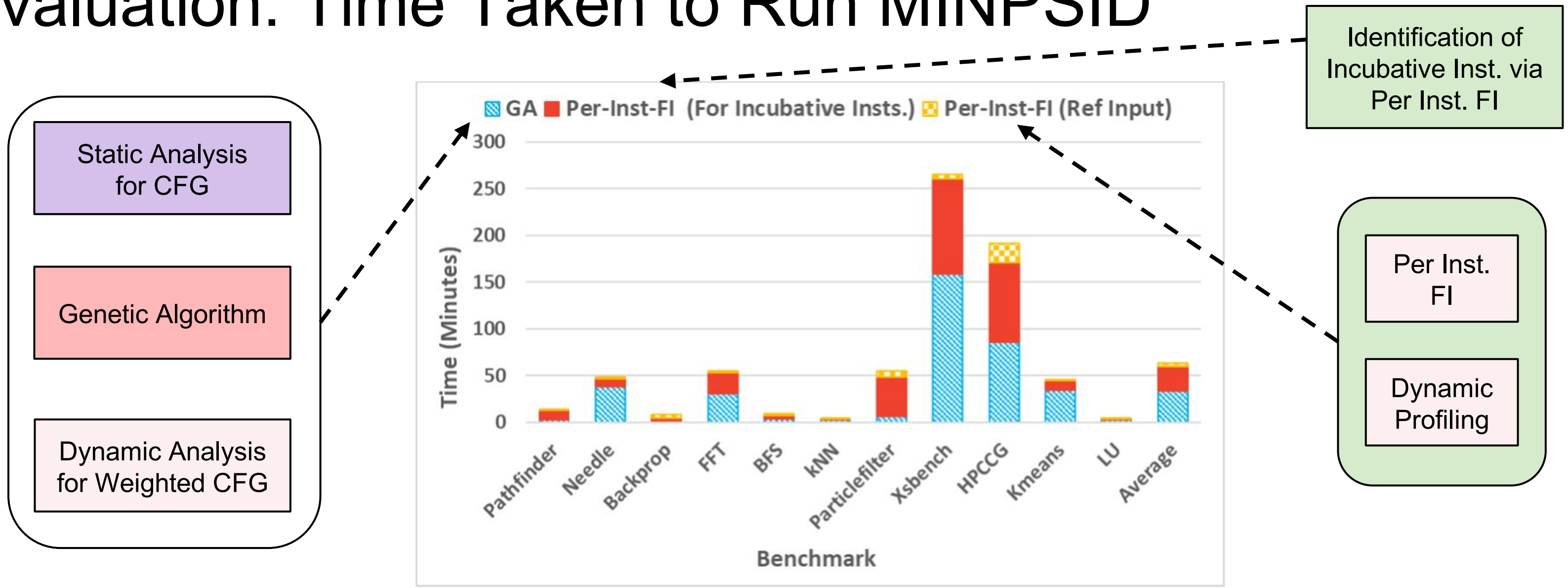
Evaluation: Finding Incubative Instructions

Random Fuzzer: Genetic algorithm with random mutation (no fitness function).



- Input search engine can identify **45.60%** more incubative instructions compared with a random fuzzer, and those more identified incubative instructions account for additional **34%** loss of SDC coverage.

Evaluation: Time Taken to Run MINPSID



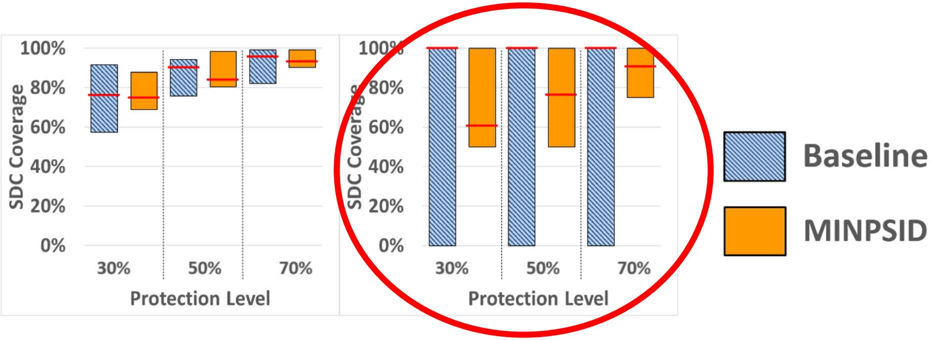
- On average, MINPSID takes **63.71** mins to finish the entire workflow.
  - Input search engine: 0.56 mins (Backprop) ~ 158.97 mins (Xsbench)
  - Per-Inst-FI (ICB. Insts): 0.88 mins (kNN) ~ 101.25 mins (Xsbench)
  - Per-Inst-FI (Ref. Input): 0.20 mins (Pathfinder) ~ 21.08 mins (HPCCG)

One time cost!

Case Study: MINPSID with Real-World Inputs

BFS - Top 30 real-world graphs from KONECT.

Kmeans - Top 10 clustering competition datasets from kaggle.`



(a) BFS (b) Kmeans

- The results are inline with what are messuared under randomly generated inputs:
  - Decreasing the SDC coverage variation by **54.77%**.
  - Reducing **85.44%** loss SDC coverage.
  - Only **16.67%** inputs lead to the loss of SDC coverage (**65.56%** in baseline SID).

Summary

SC'22

Obtained all 3 badges

- Input variation leads to the loss of SDC coverage of programs under SID protection.
- Incubative instructions account for the loss of SDC coverage.
- MINPSID can efficiently identify incubative instructions, and hence harden SID across multiple program inputs.
- MINPSID also works efficiently for programs under the real-world inputs.
- Open Source: https://github.com/hyfshishen/SC22-MINPSID

Yafan Huang
University of Iowa
yafan-huang@uiowa.edu
https://hyfshishen.github.io

This research was supported in part by the National Science Foundation (NSF) under Grant No. 2211538 and 2211539, and the U.S. Department of Energy, Office of Science under contract DE-AC02-06CH11357. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, the U.S. Department of Energy, or Baidu.