ISSRE 2023

Florence, Italy

Characterizing Runtime Performance Variation in Error Detection by Duplicating Instructions

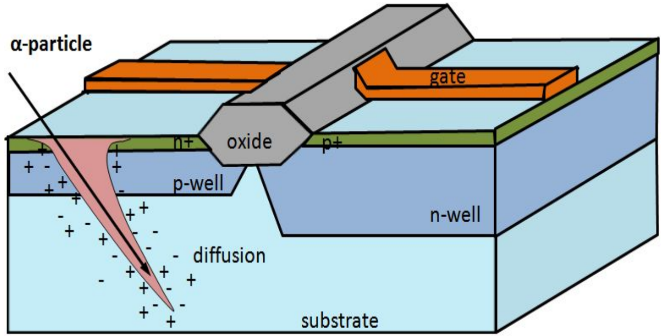
Yafan Huang*, Zhengyang He*, Lingda Li, Guanpeng Li

THE UNIVERSITY OF IOWA ORGANIZED FEB. 25TH 1847.

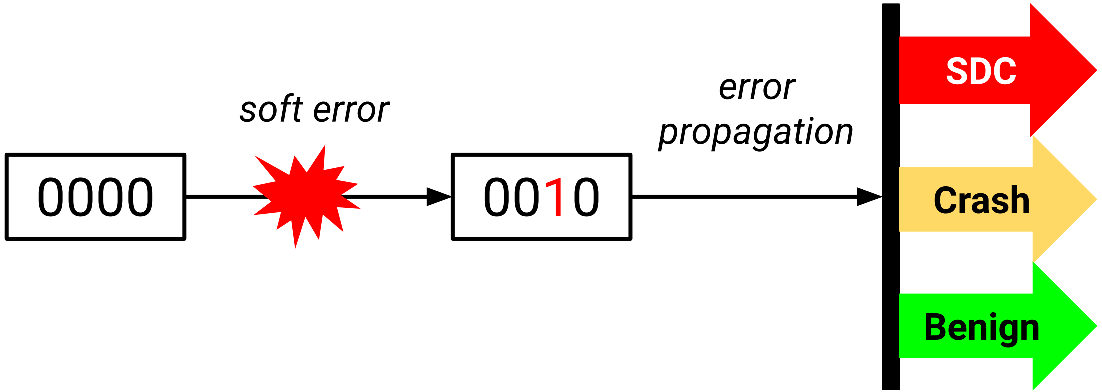
Brookhaven National Laboratory

Soft Error

- **Soft error** is becoming **prevalent** in modern processors.
- Soft error may lead to severe **failure outcomes**, hence should be mitigated.



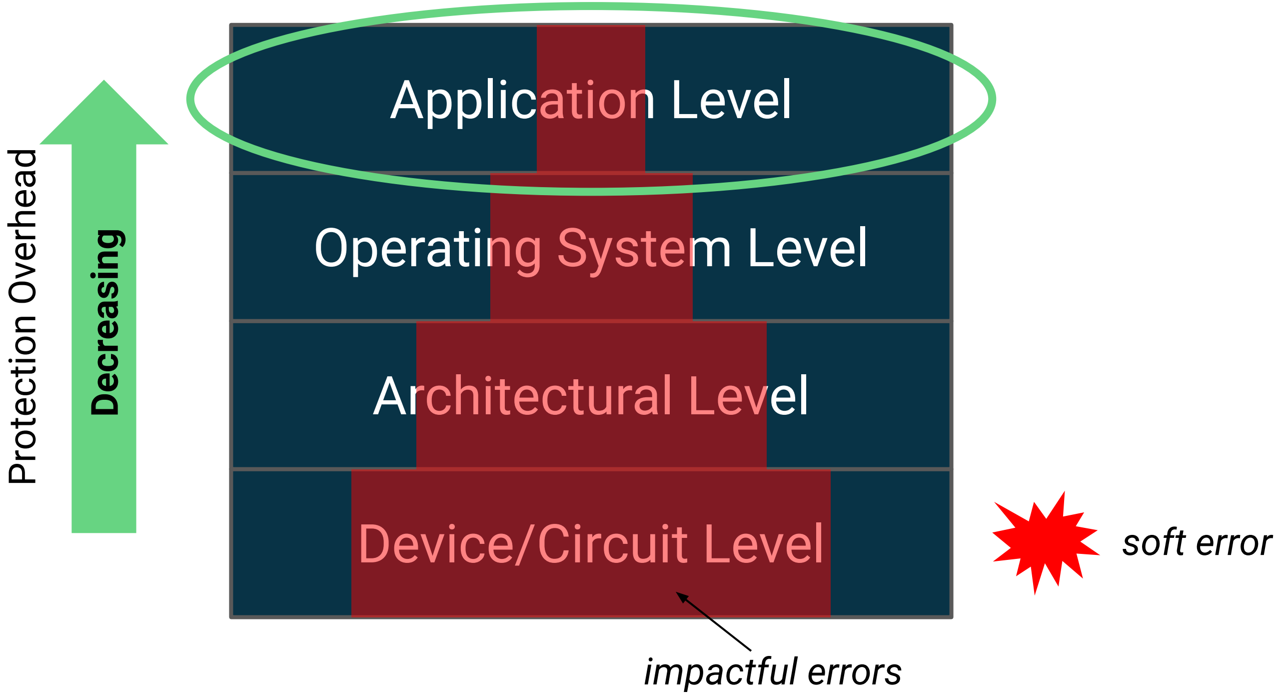
[1]



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

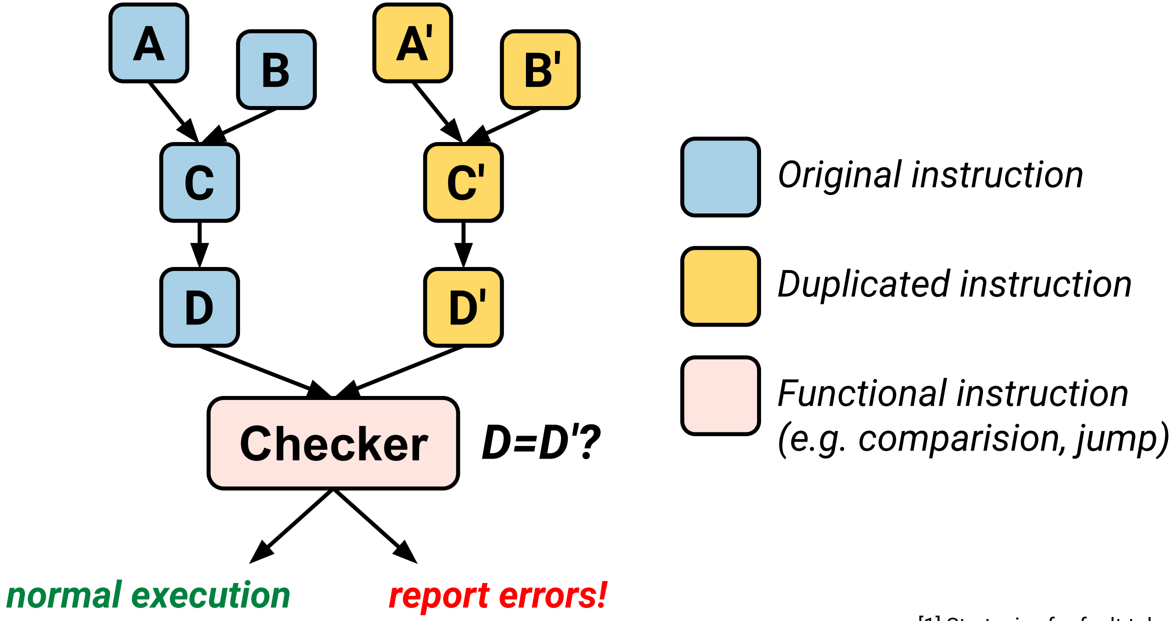
Software Solutions

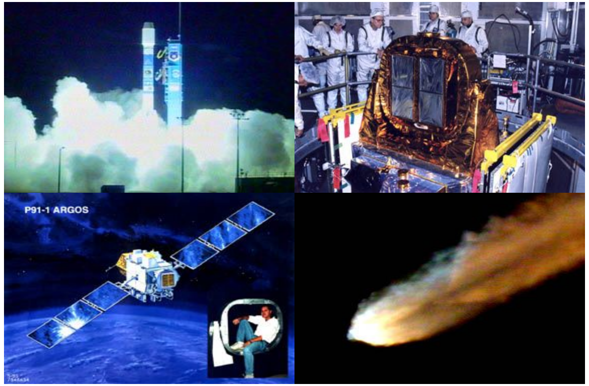
- **Software solution** is more flexible and cost-effective.



Error Detection by Duplicating Instructions (EDDI)

- **EDDI** duplicates instructions at compile-time and detects errors at run-time.
- Compiler-level transformation, hence **program-agnostic**.



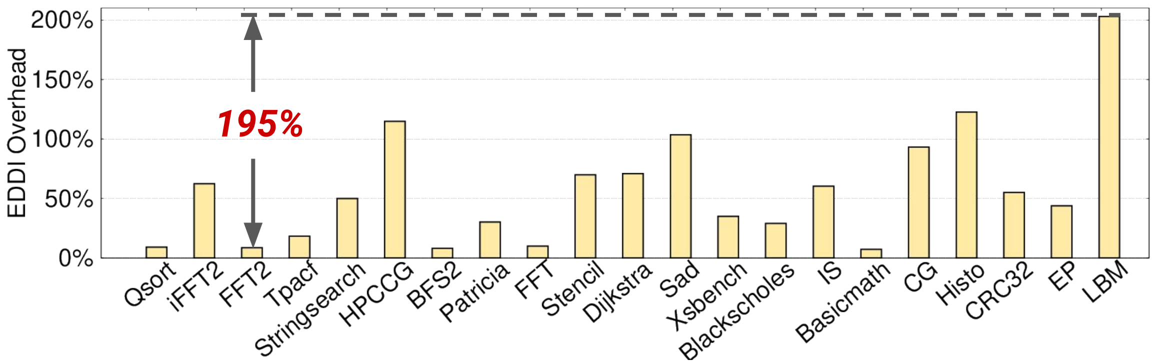


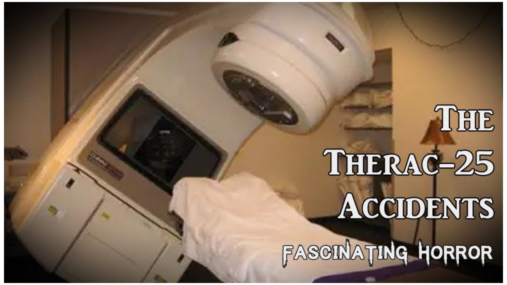
Argos Project[1]

[1] Strategies for fault-tolerant, space-based computing: Lessons learned from the ARGOS testbed

Motivation: Runtime Performance Variation in EDDI

- EDDI runtime performance varies a lot across different programs.
  - From **8%** on FFT2 to **203%** on LBM benchmarks.
- Understanding performance variation is important to real-time systems.
  - **Therac-25 accidents**[1] 1985~1987, due to unscheduled subcomponents.
- ***This is the first work for studying EDDI runtime performance.***





[1] An investigation of the therac-25 accidents

Goal

- Goal: **characterize and understand** performance variation of EDDI.
  - **G1**: Identify root-causes that affect EDDI the most.

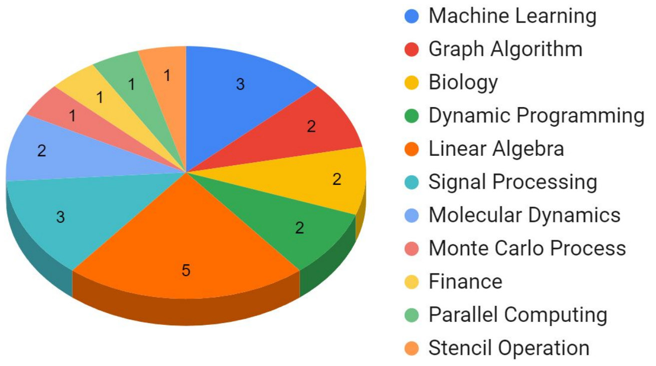
    ↳ A comprehensive correlation study.

  - **G2**: Assist system-designers to develop safe and performant EDDI.

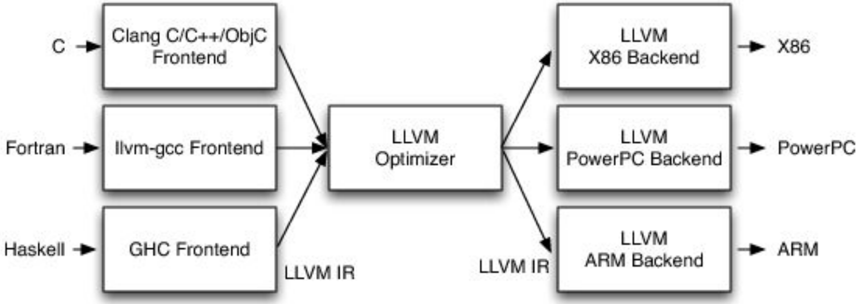
    ↳ Two techniques: **FuzzyB** and **Celer**.

Experimental Setups

- Platform
  - Ubuntu 20.04 OS.
  - Intel Core i7-10700 processor.
  - 64 GB RAM.
- Benchmarks
  - 22 open-source benchmarks.
- EDDI Implementation
  - LLVM transformation passes.
  - Full duplication.



Benchmark Application Domains



LLVM Compiler Infrastructure[1]

[1] https://llvm.org/

Correlation Study: Methodology

- Correlation Study
  - How **strong** two arrays **are related** to each other.
  - EDDI runtime performance and target factor.
  - [-1, 1], where |*cor*| > 0.3 can be seen as correlated[1].

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

Pearson Correlation Coefficient

- Target Factors
  - 10 **program**-level factors
  - 12 **architecture**-level factors
- Profiling Tools
  - Architecture-level: Linux Perf
  - Program-level: LLVM passes

Program-level factors	***Architecture-level factors***
dynamic instructions	*L1 dcache loads*
fp binary operators	*L1 icache load misses*
int binary operators	*L2 cache instruction hits*
logical binary operators	*L2 cache instruction misses*
basic blocks	*LLC loads*
Branch	*LLC stores*
…	…

[1] Scatterplots and correlation, 2017

Correlation Study: Results

Factor	Category	Description	Measurement tool	Correlation
Dynamic-instructions	Instruction	Dynamic instruction count.	Compiler instrumentation	0.69
Standard-binary-operators	Instruction	Standard binary instruction (e.g FAdd, Add) count.	Compiler instrumentation	0.44
Floating-point-binary-operators	Instruction	Floating-point binary instruction (e.g FAdd) count.	Compiler instrumentation	0.39
Integer-binary-operators	Instruction	Integer binary instruction (e.g Add) count.	Compiler instrumentation	0.23
Logical-operators	Instruction	[illegible]	Compiler instrumentation	-0.02
Cast-operators	Instruction	[illegible]	Compiler instrumentation	0.44
Cmp-operators	Instruction	Comparison instruction (e.g Icmp) count.	Compiler instrumentation	0.04
Basic-blocks	Instruction	Dynamic basic block count.	Compiler instrumentation	0.44
Branch	Instruction	[illegible]	Linux profiler Perf	0.13
Branch-misses	Instruction	[illegible]	Linux profiler Perf	-0.22
L1-dcache-loads	Cache/memory	L1 data cache load count.	Linux profiler Perf	0.63
L1-dcache-stores	Cache/memory	L1 data cache store count.	Linux profiler Perf	0.17
L1-dcache-load-misses	Cache/memory	L1 data cache load miss count.	Linux profiler Perf	-0.11
L1-icache-load-misses	Cache/memory	L1 instruction cache miss count.	Linux profiler Perf	0.42
L2-cache-instruction-hits	Cache/memory	L2 cache instruction fetch hit count.	Linux profiler Perf	0.42
L2-cache-instruction-misses	Cache/memory	L2 cache instruction fetch miss count.	Linux profiler Perf	[illegible]
L2-cache-data-hits	Cache/memory	L2 cache data request hit count.	Linux profiler Perf	[illegible]
L2-cache-data-misses	Cache/memory	L2 cache data request miss count.	Linux profiler Perf	[illegible]
LLC-loads	Cache/memory	L3 cache load execution count.	Linux profiler Perf	[illegible]
LLC-load-misses	Cache/memory	L3 cache load miss execution count.	Linux profiler Perf	[illegible]
LLC-stores	Cache/memory	L3 cache store execution count.	Linux profiler Perf	0.05
LLC-store-misses	Cache/memory	L3 cache store miss count.	Linux profiler Perf	0.01

Instruction: Program-level

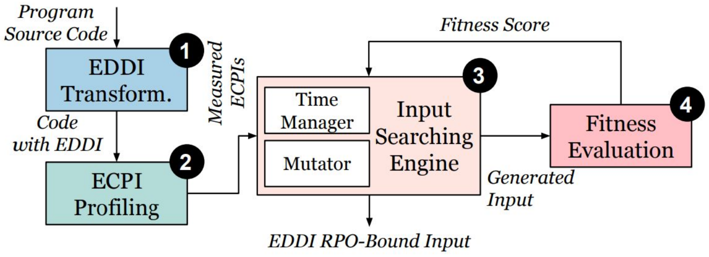
Cache/memory: Architectural

✗ *Not consider due to limited usage!*

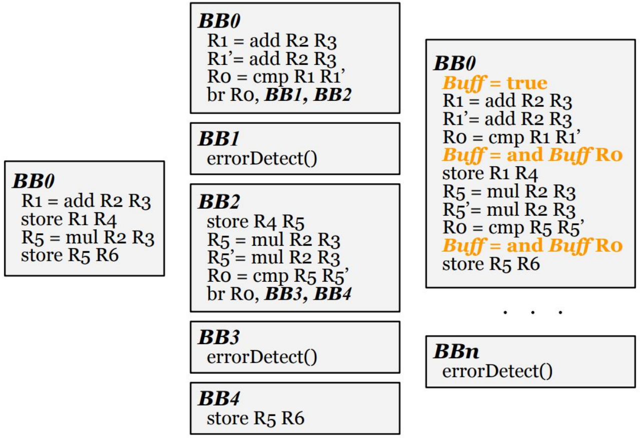
- We found 6 factors (5 + 1) that are correlated with EDDI performance variation.

Correlation Study: Summary

- 6 factors (5 + 1) that are correlated with EDDI performance variation.
- Two techniques to assist the usage of EDDI in real-world applications:
  - **FuzzyB**: bounding EDDI performance variation with the identified factors.
  - **Celer**: accelerating EDDI performance with optimized program control-flow.



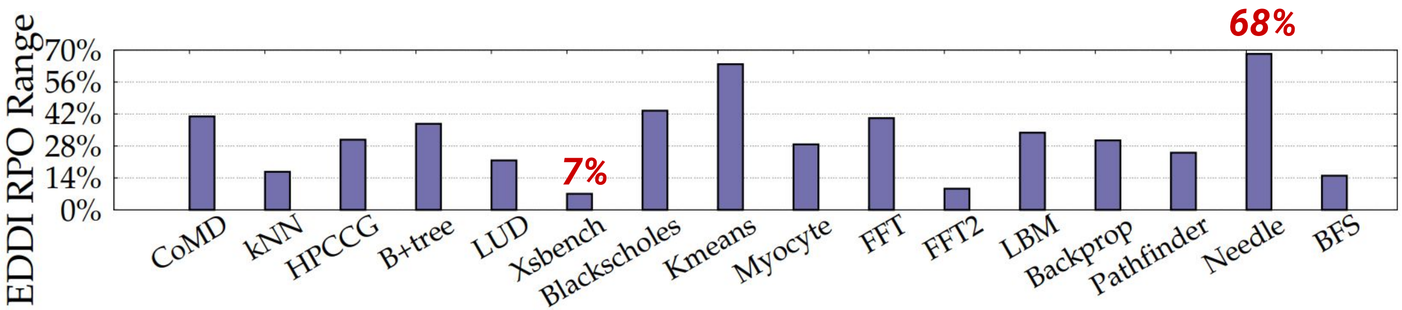
FuzzyB Workflow



Celer Example

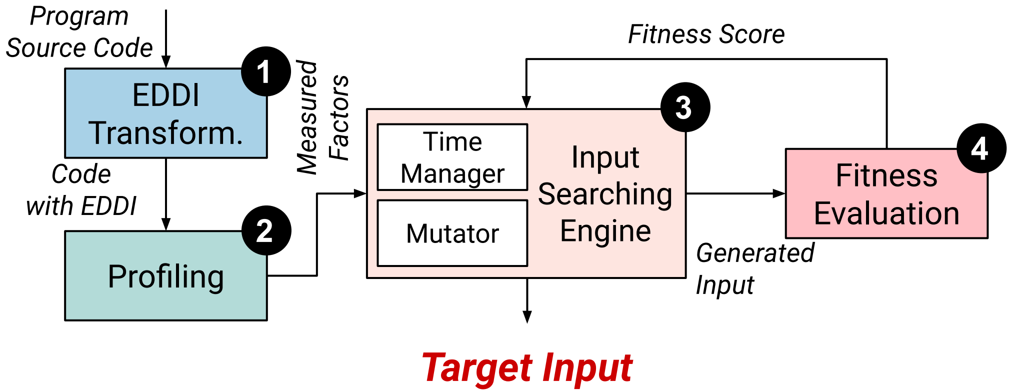
FuzzyB: Bounding EDDI Performance Variation

- EDDI performance not varies across benchmarks, but also across **inputs**.
  - From **7%** in Xsbench to **68%** in Needle.
- **Fuzzing technique** can locate input with a certain feature.
  - **6** identified factors contribute dominantly to such feature.



EDDI Performance Variation across Different Inputs

FuzzyB: Bounding EDDI Performance Variation



$$\text{Fitness}_i = \frac{1}{2}\text{RPO}_i + \frac{1}{2}\sum_{k=1}^{K}\frac{e^{\text{FC}_k}}{\sum_{j=1}^{K} e^{\text{FC}_j}}\boxed{\text{Factor}_{ki}}$$

each identified factor

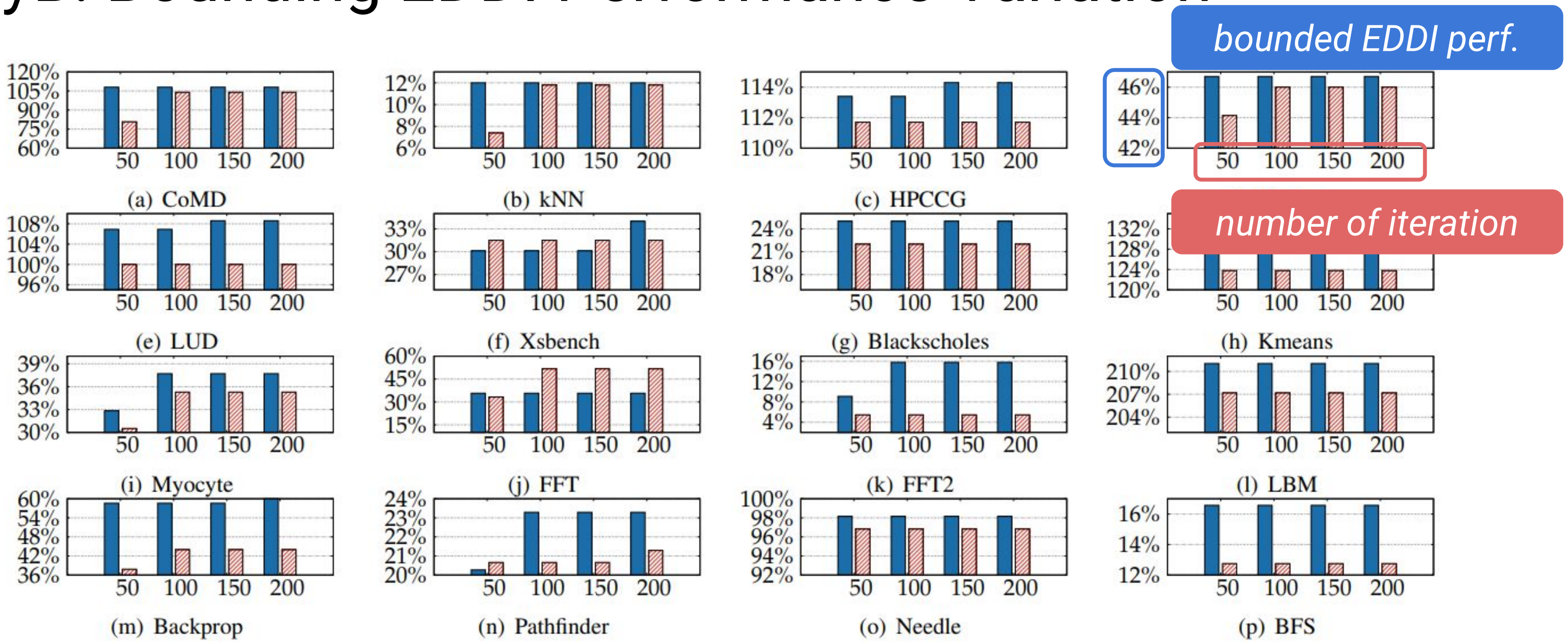
upper bound

- **Input searching engine**: a fuzzing-based technique to **bound** EDDI performance within a certain number of iterations.

guide searching engine

- **Fitness score**: 6 identified factors weighted by softmax function.

FuzzyB: Bounding EDDI Performance Variation



*Bounding EDDI performance with FuzzyB (**left**) and Random Fuzzer (**right**)*

- FuzzyB bound higher EDDI runtime performance with fewer iterations.

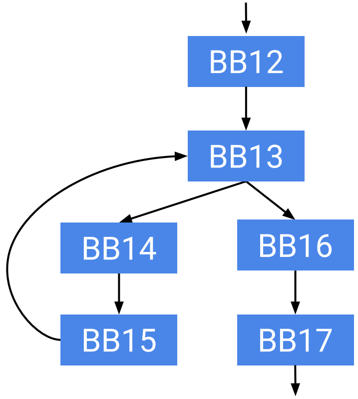
Celer: Accelerating EDDI Performance

- Can we let EDDI run faster?
- 6 identified factors:
  - dynamic instructions
  - stdbin operators
  - fp operators
  - cast operators
  - basic blocks
  - L1 dcache loads

program-specific (dynamic instructions, stdbin operators, fp operators, cast operators)

basic blocks → ***Our target!***

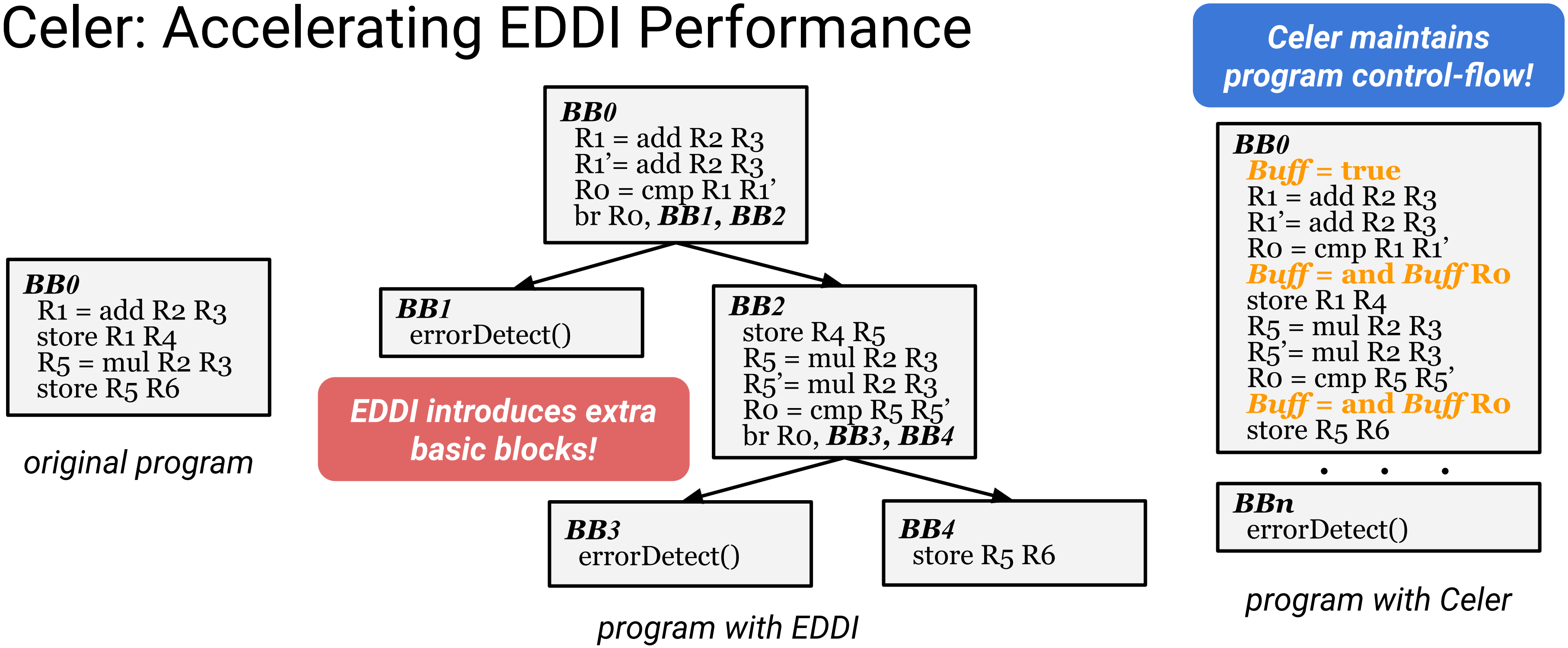
L1 dcache loads → *invisible to developer*



program control-flow and basic blocks

- Accelerating EDDI runtime performance by **reducing number of basic blocks.**

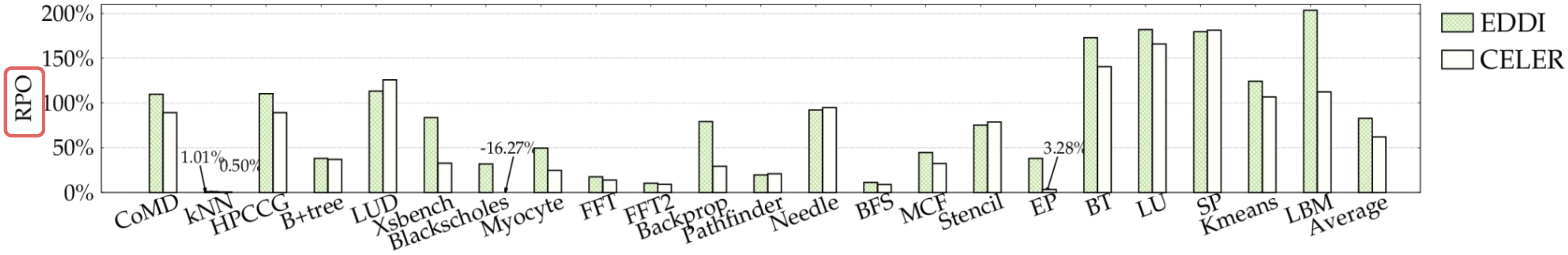
Celer: Accelerating EDDI Performance



- Celer does not increase basic block via **simplifying control-flow** with a buffer.
- Celer is a variant of EDDI without any losses of soft error detection effectiveness.

Celer: Accelerating EDDI Performance

RPO denotes "runtime performance overhead"



EDDI Runtime Performance between EDDI and Celer

- Celer reduce more than **99%** extra dynamic basic blocks in EDDI.
- On average, Celer improve EDDI runtime performance by **25%**.

Summary

- EDDI runtime performance varies across both programs and inputs.
- 6 factors dominantly contribute to such variations.
- FuzzyB efficiently bound EDDI runtime performance across different inputs.
- Celer can accelerate EDDI runtime performance by 25%.
- Open source: https://github.com/hyfshishen/ISSRE23-FUZZYB-CELER

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

THE UNIVERSITY OF IOWA ORGANIZED FEB. 25TH 1847.

Brookhaven National Laboratory