A Fast Low-Level Error Detection Technique

Zhengyang He, Hui Xu, Guanpeng Li

Reza Adinepour

Amirkabir University of Technology
(Tehran Polytechnic)

Computer Engineering Department
January 13, 2025

Agenda

Problem & Solutions Overview

- **Problem:** Transient hardware faults (soft errors) due to shrinking transistor sizes and operating voltages.
- **Impact:** Soft errors can cause Silent Data Corruptions (SDCs), compromising system dependability.
- **Solutions:**
  1. Traditional: Hardware-based methods such as:
     - voltage guard bands
     - redundancy

     have high overhead in performance and energy consumption.
  2. Software-Based: Error Detection by Duplicating Instructions (EDDI)

     has been proposed as a flexible, resource-efficient alternative.

EDDI Methods

- **EDDI:** Duplicates instructions at compile time and checks for mismatches at runtime.

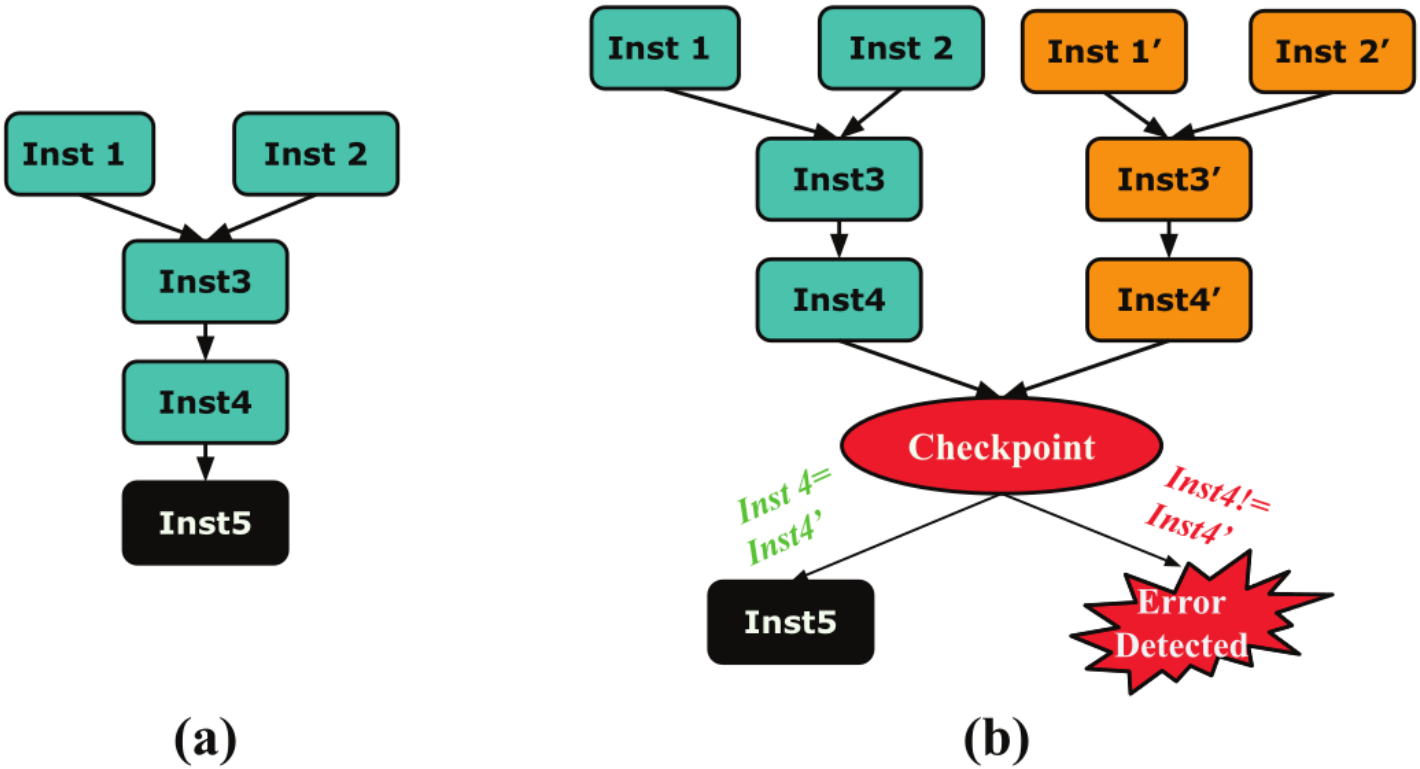


Figure: High-level idea of EDDI

EDDI Methods (Cont.)

- **Existing EDDI Methods:**
  1. Mostly at IR level

  reduced fault coverage when tested at the assembly level.
- **Problem with IR-Level EDDI:**
  - Fault coverage gaps at IR level.
  - Reduced effectiveness when evaluated at assembly level.
  - Underestimated error detection at lower levels.
  - Need for assembly-level implementation for better fault protection.

IR Code Example Using EDDI

```
// High-level C code
int add(int a, int b) {
    return a + b;
}
```

```
define i32 @add(i32 %a, i32 %b) {
entry:
  %a.addr = alloca i32, align 4
  %b.addr = alloca i32, align 4
  store i32 %a, i32* %a.addr, align 4
  store i32 %b, i32* %b.addr, align 4
;Duplicate instruction
  %0 = load i32, i32* %a.addr, align 4
  %1 = load i32, i32* %a.addr, align 4
;Duplicate instruction
  %2 = load i32, i32* %b.addr, align 4
```

Figure: (a)

```
  %3 = load i32, i32* %b.addr, align 4
;Duplicate instruction
  %add = add nsw i32 %0, %1
  %add2 = add nsw i32 %2, %3
;Check the results
  %cmp = icmp eq i8** %add, %add2
  br i1 %cmp, label %4, label %checkBb

checkBb:
  call void @check_flag()
  br label %4

<label>:4
  ret i32 %add
}
```

Figure: (b)

Main Contribution

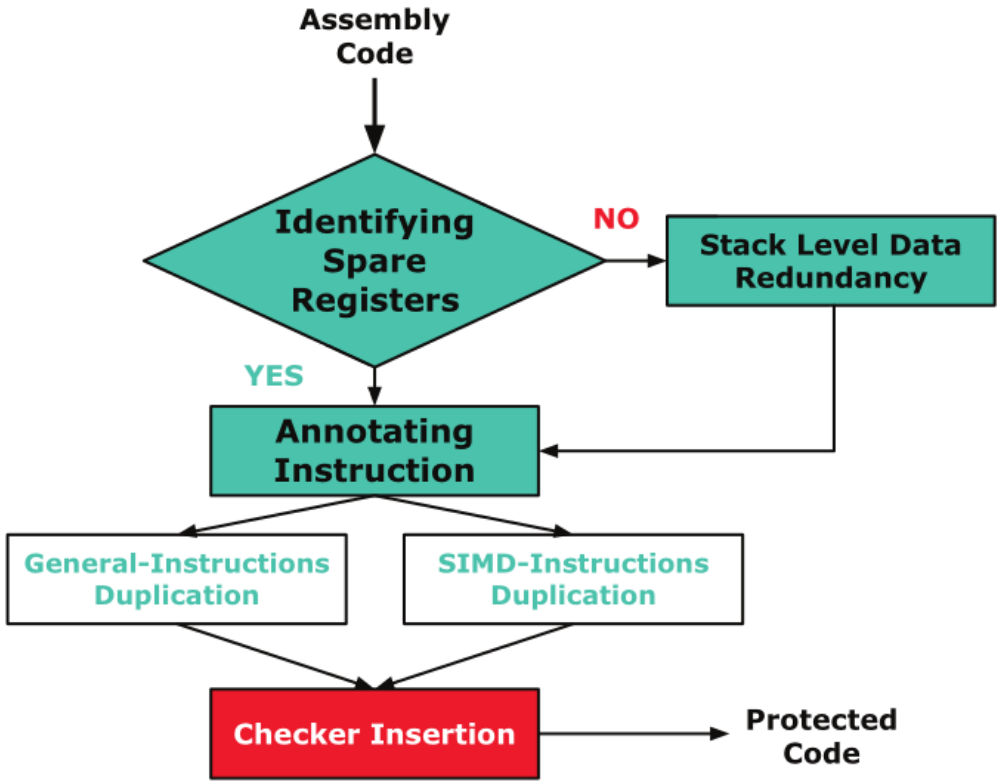
1. **Proposed Solution:**
   - FERRUM: Optimized assembly-level EDDI.
   - Enhancements: Utilizes SIMD and compiler optimizations.
   - Improves: Fault coverage and performance.
2. **Key Findings & Results:**
   - 28% gap in fault coverage (IR-level vs. assembly-level).
   - 100% fault coverage with FERRUM at assembly level.
   - 52% reduction in runtime overhead with FERRUM, no loss in fault coverage.

Background

1. **Focus on single bit-flip transient faults in:**
   - Processor computing components
   - Pipeline stages
   - Arithmetic components
   - Load/store units

   Do not consider faults in the memory or caches, as we assume they have already been protected by ECC (Error Correcting Code).
2. **Fault Simulation:** Assembly-level fault injection; beam testing infeasible.
3. **EDDI:** Instruction duplication, runtime comparison.
4. **Platform:** x86 ISA (other platforms for future work).

High-Level Design



- Scan registers (general-purpose, SIMD); identify spare registers.
- Annotate instructions for SIMD compatibility.
- Duplicate instructions; use SIMD or general-purpose registers.

Components

1. **Static Code Analysis**
   - Identify spare registers (general-purpose: 2, SIMD: 4 XMM).
   - Annotate instructions (SIMD-enabled or general).
2. **Duplication for General Instructions**
   - Duplicate instructions; use spare registers or deferred detection for comparisons (e.g., `rflag`).
3. **Duplication for SIMD-Enabled Instructions**
   - Use SIMD registers (e.g., XMM, YMM) for bulk comparison.
   - Leverage architecture-specific features (e.g., ZMM on Intel CPUs).

Example1

```
.LBB0_3:
...
movslq  %ecx, %r10
movslq  %ecx, %rcx #original instruction
xorq    %rcx, %r10
jne exit_function
...
```

Figure: Protection of GENERAL-INSTRUCTIONS (`movslq`)

Example2

```
BB1:
movq        -24(%rbp), %xmm0
movq        -24(%rbp), %rax #original Ins
movq        %rax, %xmm1
pinsrq      $1, 8(%rax), %xmm0
movq        8(%rax), %rdi   #original Ins
pinsrq      $1, %rdi, %xmm1
...
movq        -24(%rbp), %xmm2
movq        -24(%rbp), %rax #original Ins
movq        %rax, %xmm3
pinsrq      $1, 16(%rax), %xmm2
movq        16(%rax), %rdi  #original Ins
pinsrq      $1, %rdi, %xmm3
vinserti128 $1, %xmm2, %ymm0, %ymm0
vinserti128 $1, %xmm3, %ymm1, %ymm1
vpxor       %ymm1, %ymm0, %ymm0
vptest      %ymm0, %ymm0
jne exit_function
...
```

Figure: FERRUM using SIMD capability

Experimental Setup

Table: Details of Benchmarks

Benchmark	**Suite**	**Domain**
Backprop	Rodinia	Machine Learning
BFS	Rodinia	Graph Algorithm
Pathfinder	Rodinia	Dynamic Programming
LUD	Rodinia	Linear Algebra
Needle	Rodinia	Dynamic Programming
kNN	Rodinia	Machine Learning
kmeans	Rodinia	Data Mining
Particlefilter	Rodinia	Noise estimator

- Platform: Ubuntu 20.04, Intel Xeon (x86-64), 64GB RAM.

Fault Injection Methodology

1. **Single bit-flip faults** injected at assembly level.
2. **1000 random faults** injected per benchmark.
1. **Metrics:**
   - SDC Coverage: Measures reduction in Silent Data Corruptions.
   - Runtime Overhead: Measures performance impact.
   - FERRUM Execution Time: Compile-time overhead.

SDC Coverage

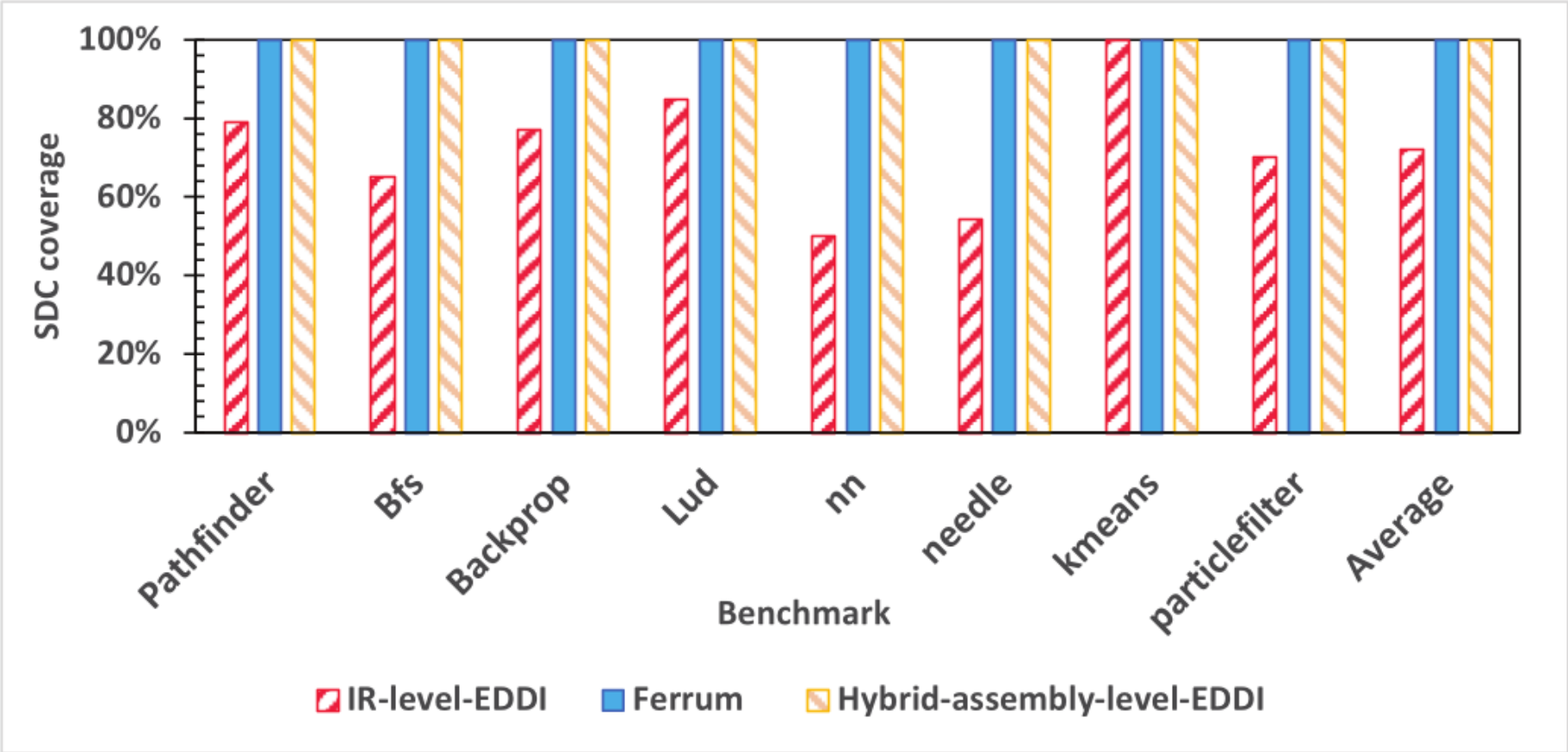


Figure: SDC coverage measured

Runtime Performance Overhead

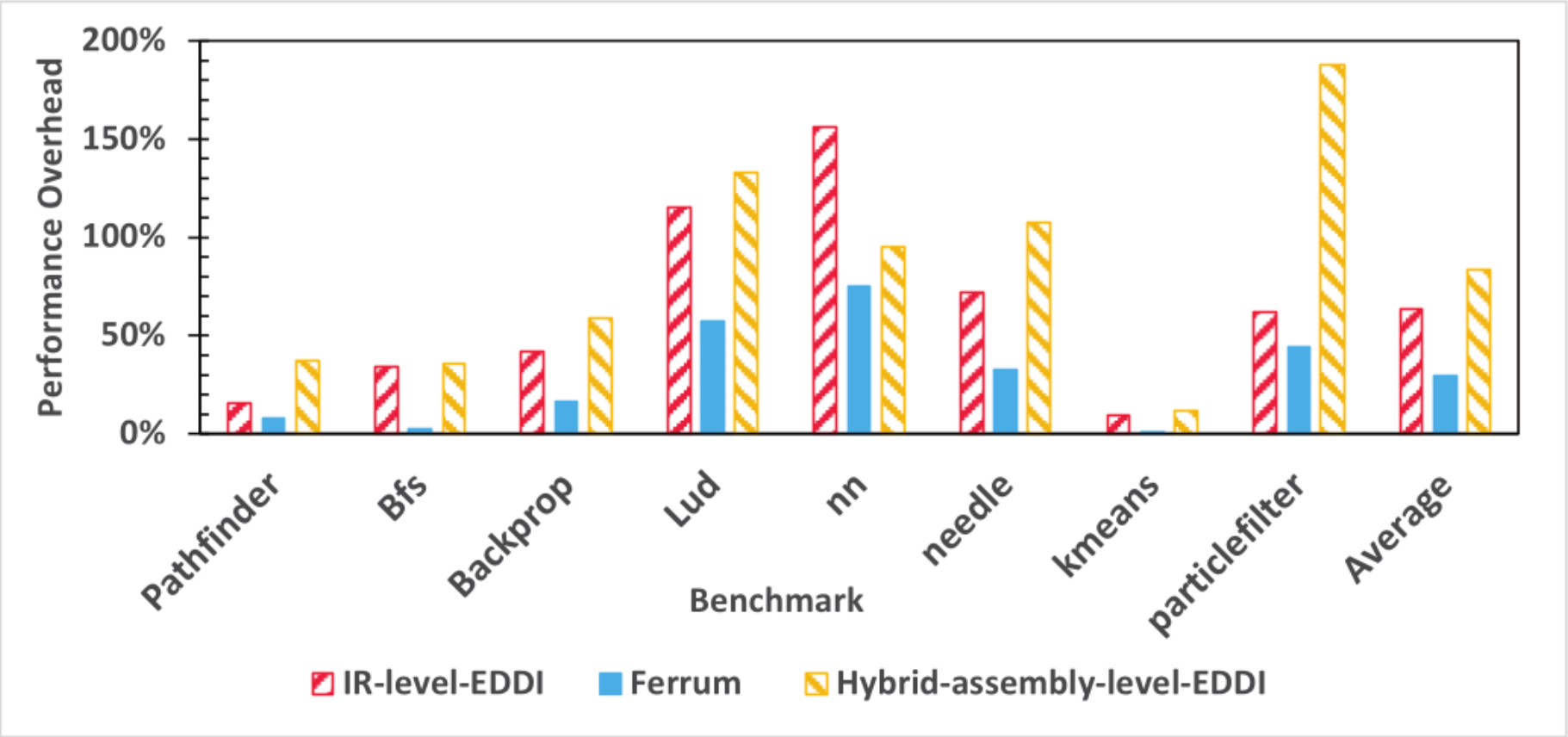


Figure: Performance overhead measured

Execution Time

1. Average: 0.117 seconds.
2. Max: 0.196 seconds.
3. Min: 0.089 seconds (BFS).

References

Zhengyang He, Hui Xu, Guanpeng Li (2024)
A Fast Low-Level Error Detection Technique
2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), University of Iowa, Iowa City, IA, USA; Fudan University, Shanghai, China.

The End

Questions? Comments?

adinepour@aut.ac.ir