Effect of Compiler Optimizations on the Error Resilience of Soft Computing Applications

Anna Thomas,
Jacques Clapauch,
Karthik Pattabiraman

University of British Columbia (UBC)

Soft Computing Applications

- Applications in AI, multimedia processing
- Expected to dominate future workloads [Dubey'07]
- Tolerate many kinds of faults in data – Error Resilient

Original image (left) versus faulty image from JPEG decoder

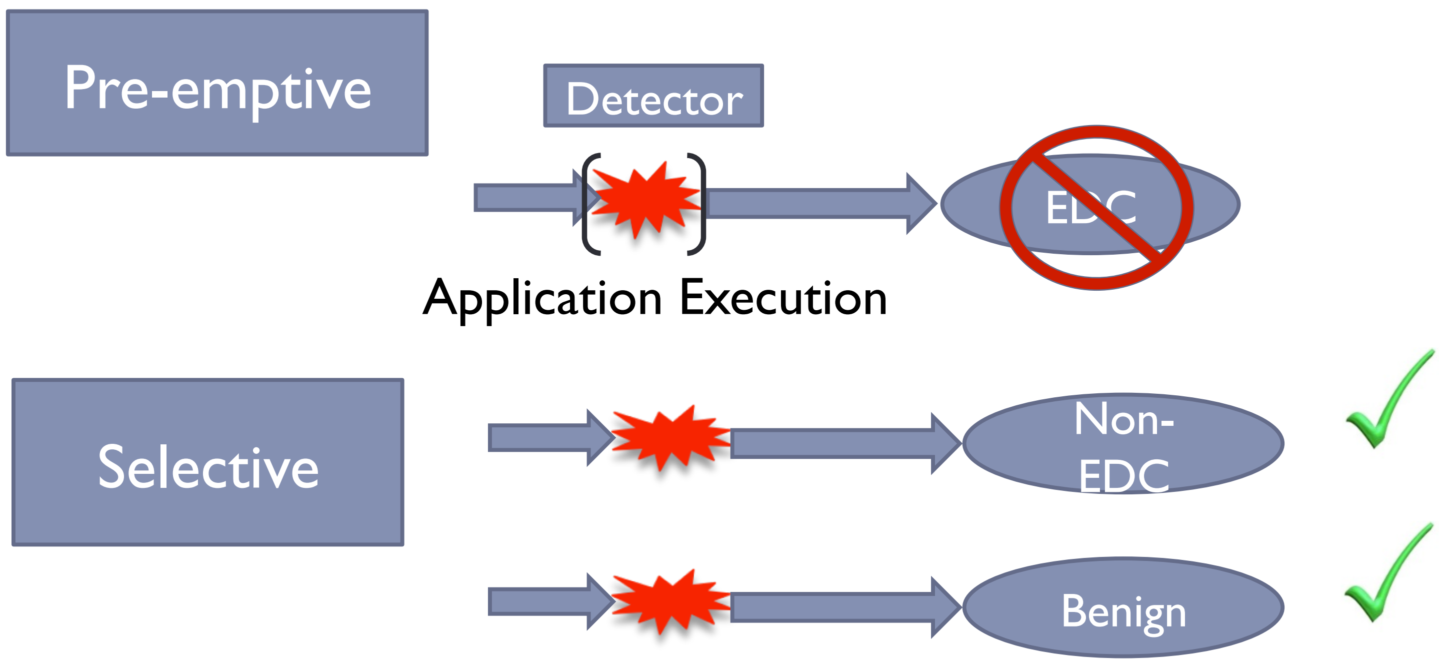
Egregious Data Corruptions (EDCs)

- Large or unacceptable deviation in output
- Based on fidelity metric (e.g., PSNR < 30)

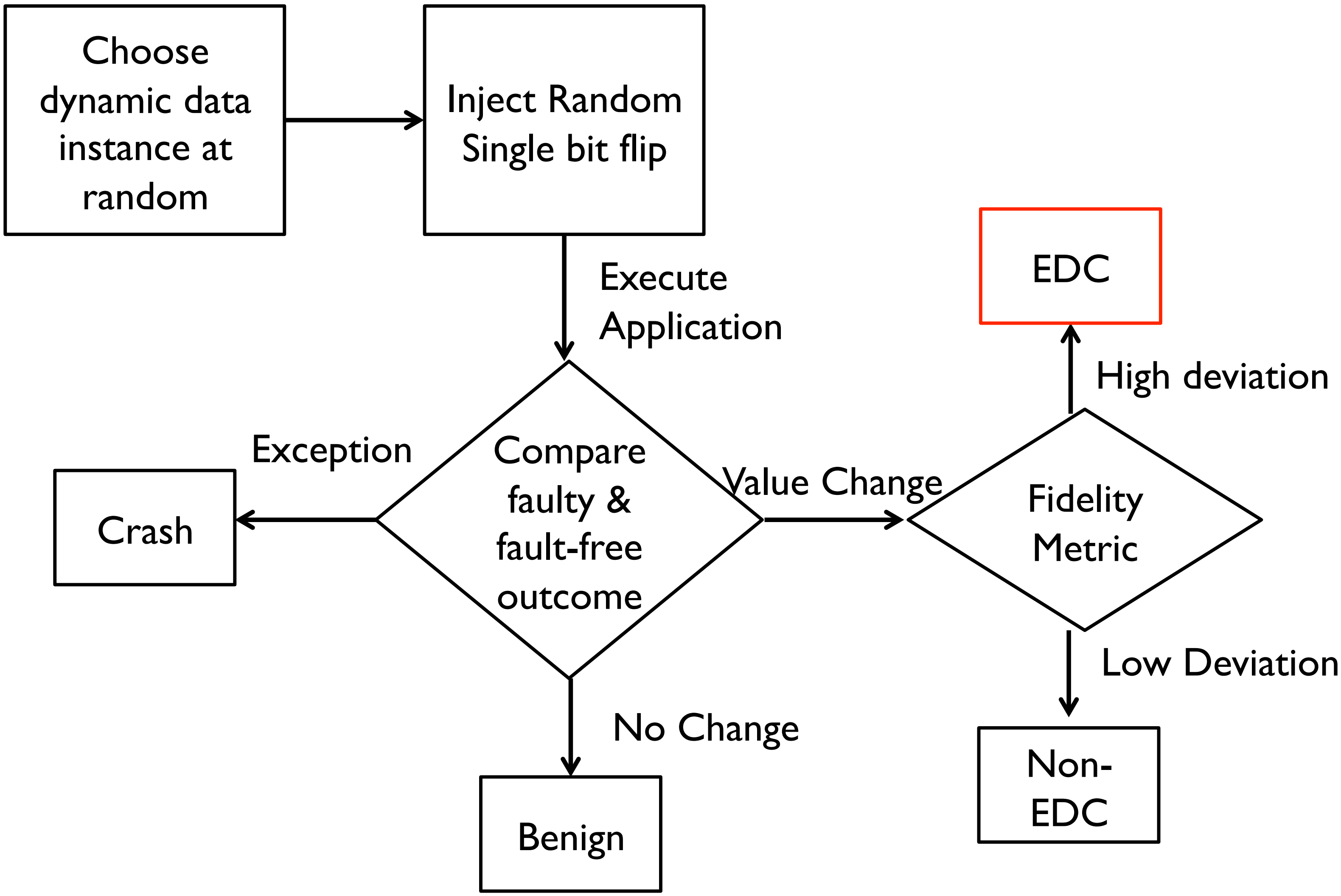
EDC image (PSNR of 11.37) of JPEG vs Non-EDC image (PSNR of 44.79)

Goal

- Detect EDC causing faults



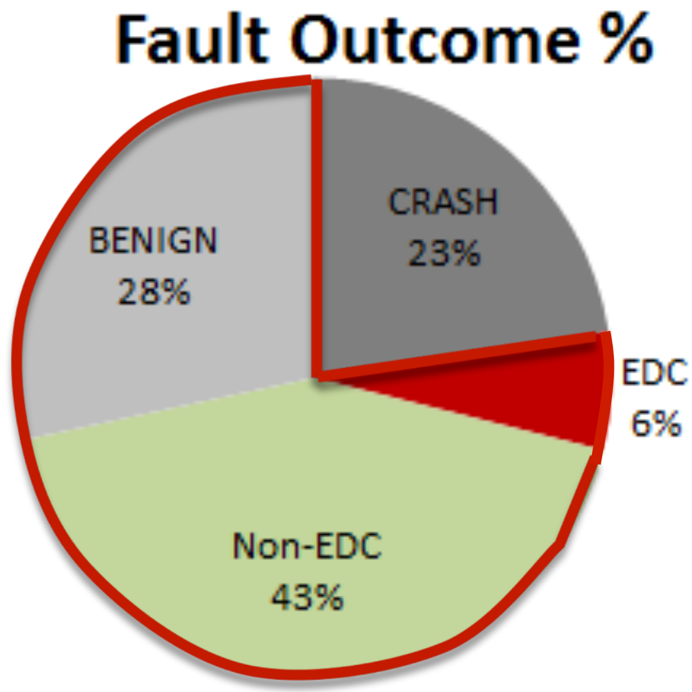
Preliminary Fault Injection Study



Why focus on EDC Causing Faults?

- ➢ Media-bench programs: Soft-computing applications

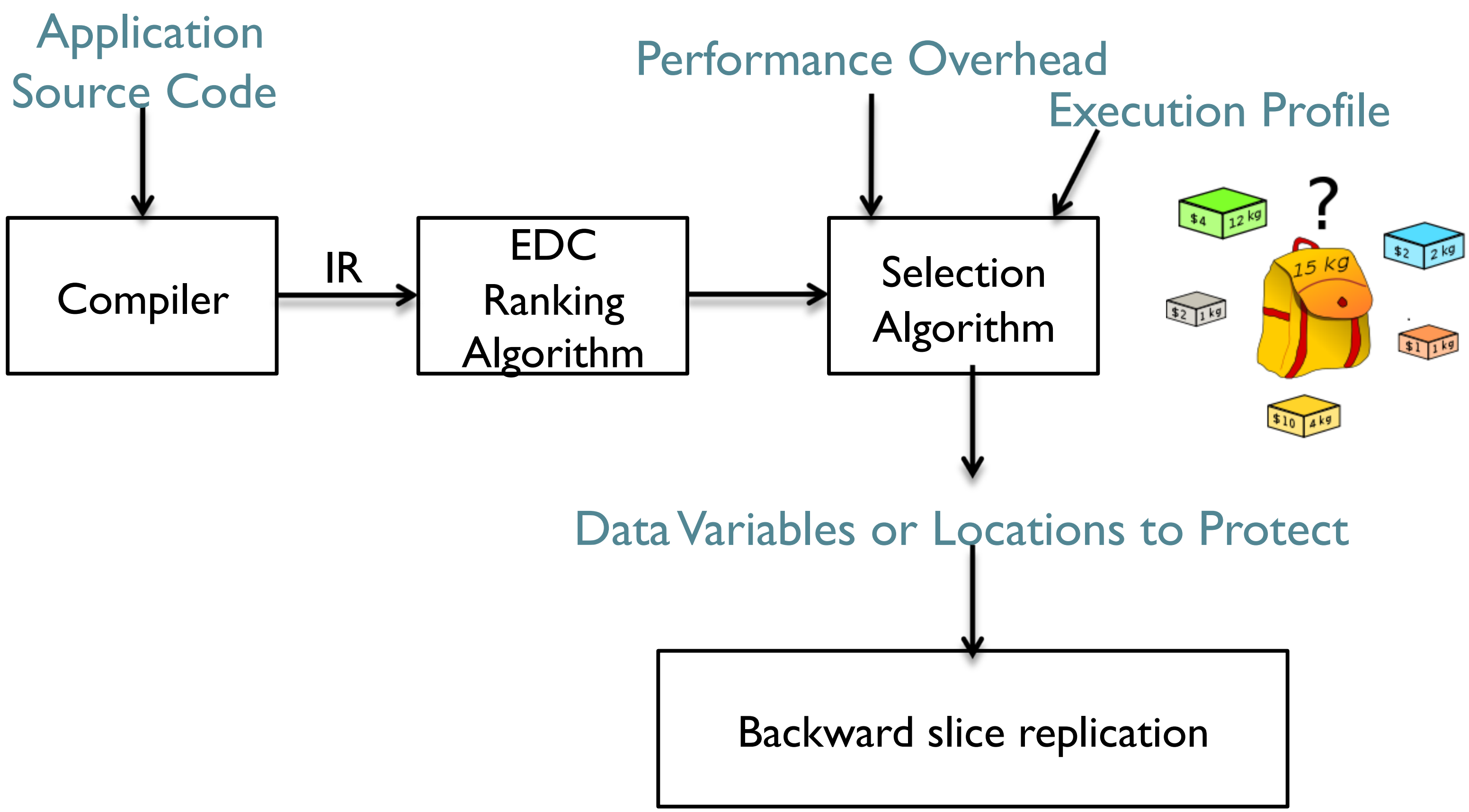
- ➢ 92% : tolerable outcomes



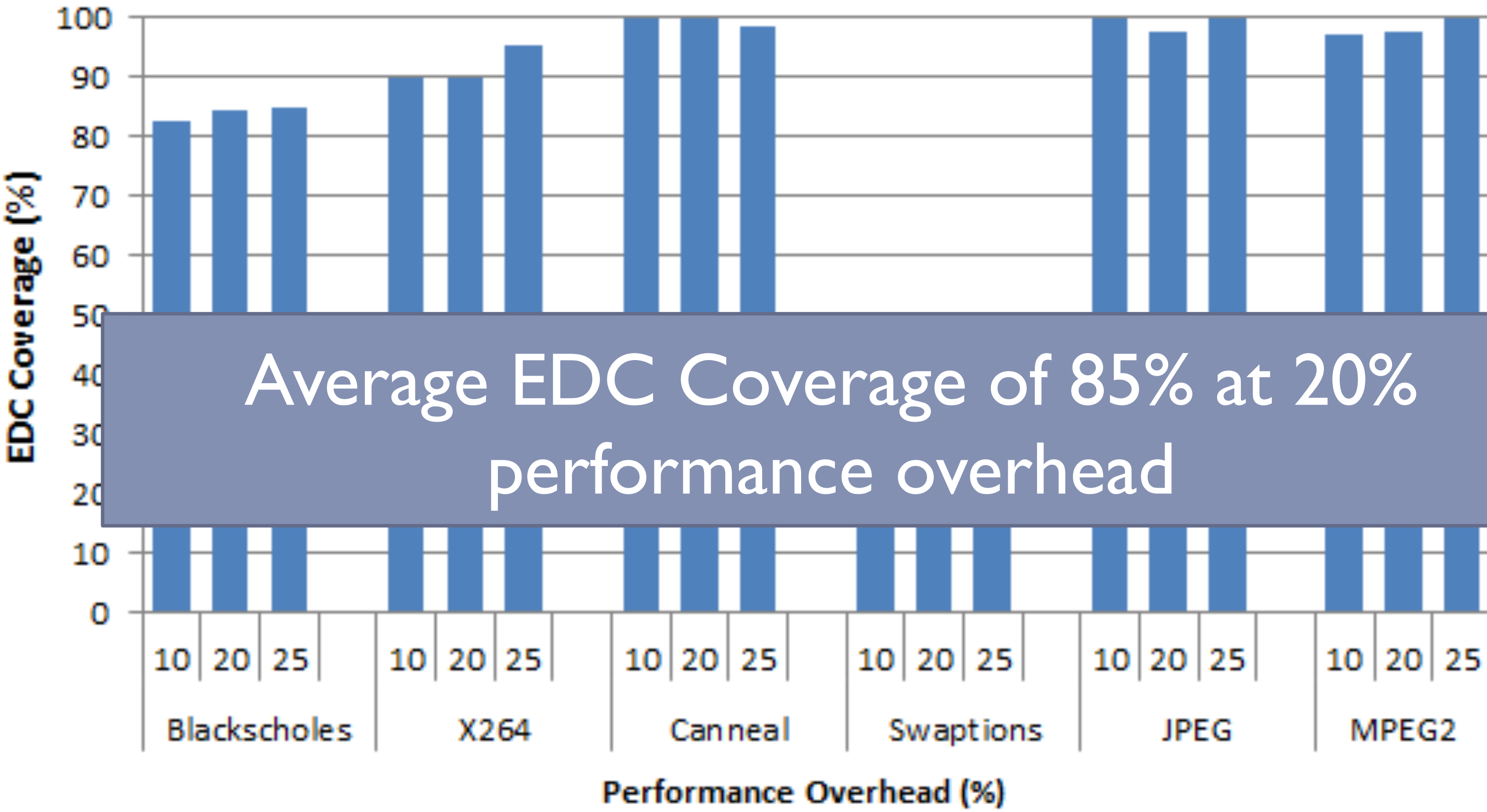
Blindly detecting all faults is wasteful !

Background

Static analysis algorithm to identify detector locations for EDCs [DSN 2013 – Anna Thomas]



Background: EDC Coverage of technique



Average EDC Coverage of 85% at 20% performance overhead

This Paper

Compiler Optimizations

Improve Performance!

Resilience

End Goal: Identify optimizations that guarantee error resilience

Performance - resilience trade off space

Outline

- Background
- Problem and Approach
- Experimental Setup and Results
- Conclusion

Problem

Optimizations affect error resilience

Affect Baseline Resilience (EDC Rate)

Affect EDC Coverage of our technique

How do optimizations affect the error resilience of soft computing applications, with our technique ?

Approach

- Identified common compiler optimizations
- Performed fault injections on unoptimized and optimized versions of soft computing apps
- Studied the effect of optimizations:
  - Baseline resilience
  - EDC coverage of our technique

Compiler Optimizations

- Identified four optimizations in LLVM compiler (of 15)
  - InstCombine
  - LICM
  - Loop Unroll
  - SCCP

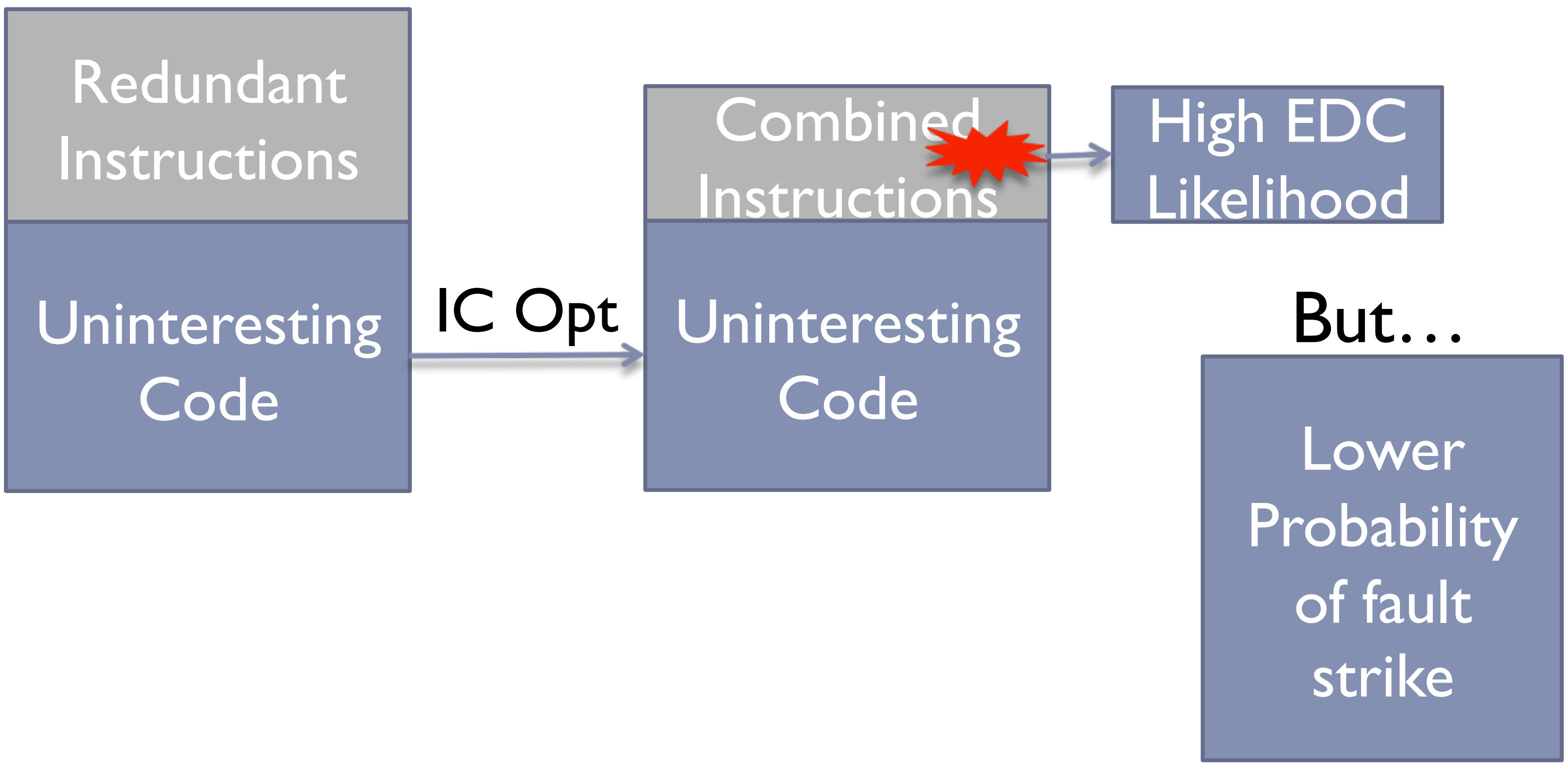
Compiler Optimizations

- Identified four optimizations in LLVM compiler (of 15)
  - **InstCombine**
  - **LICM**
  - ~~Loop Unroll~~
  - ~~SCCP~~

- Why these optimizations ?
  - Have conflicting effects on EDC rate

Compiler Optimizations

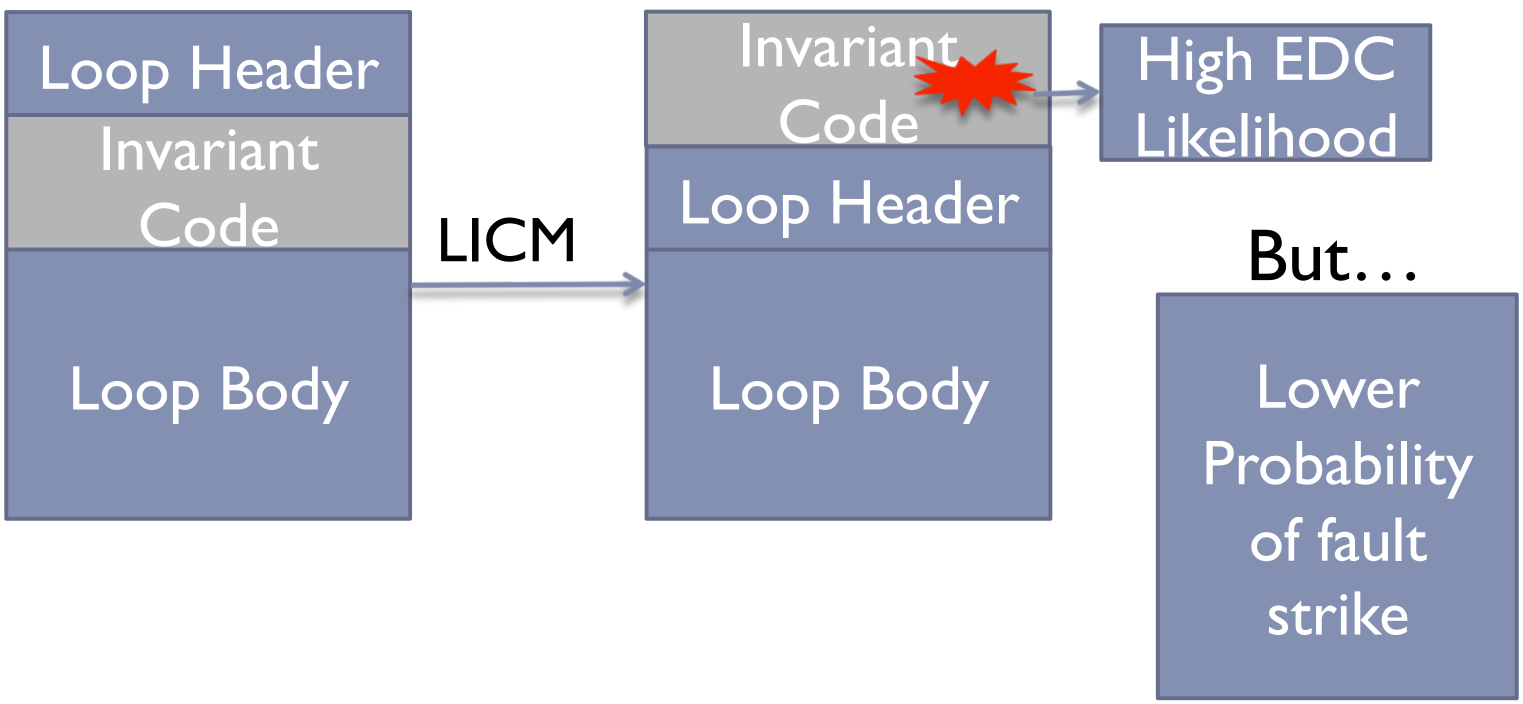
Combine Redundant Instructions: Inst-Combine



Conflicting effect on EDC rate!

Compiler Optimizations

Loop Invariant Code Motion: LICM



Conflicting effect on EDC rate!

Outline

- Background
- Problem and Approach
- Experimental Setup and Results
- Conclusion

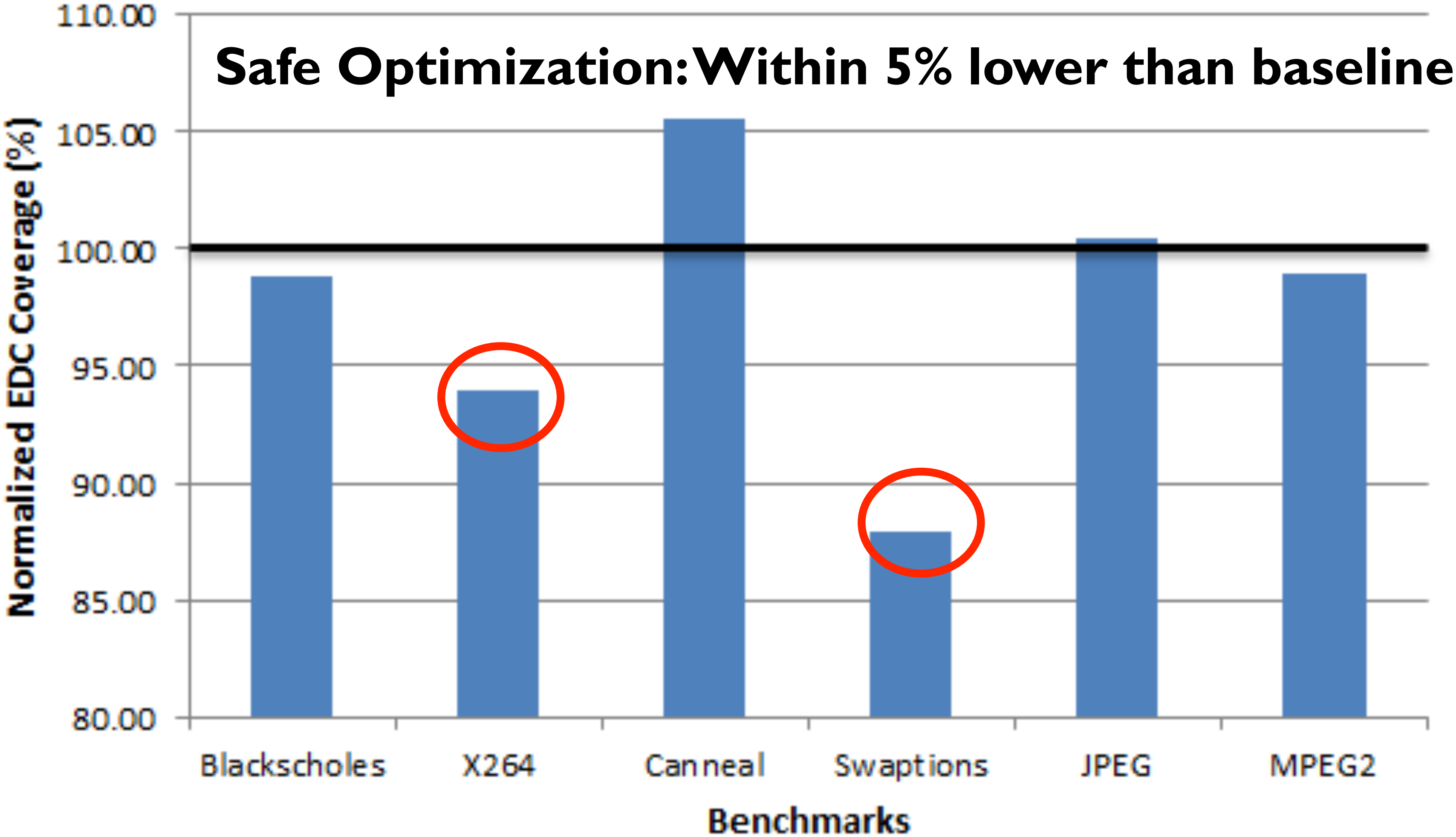
Experimental Setup

- Six Benchmarks from MediaBench and Parsec Suite
  - Fidelity Metric: PSNR, scaled distortion [Misailovic12]
  - EDC thresholds based on visual perception or 30% of SDCs
- Performed fault injections using LLFI [Thomas 13]
  - 5000 fault injections per benchmark per optimization
  - Error Bars: (± 0.8% at 95% CI)
- Measured coverage under 20% performance overhead
  - Coverage = (No of EDCs detected / total no of EDCs)*100

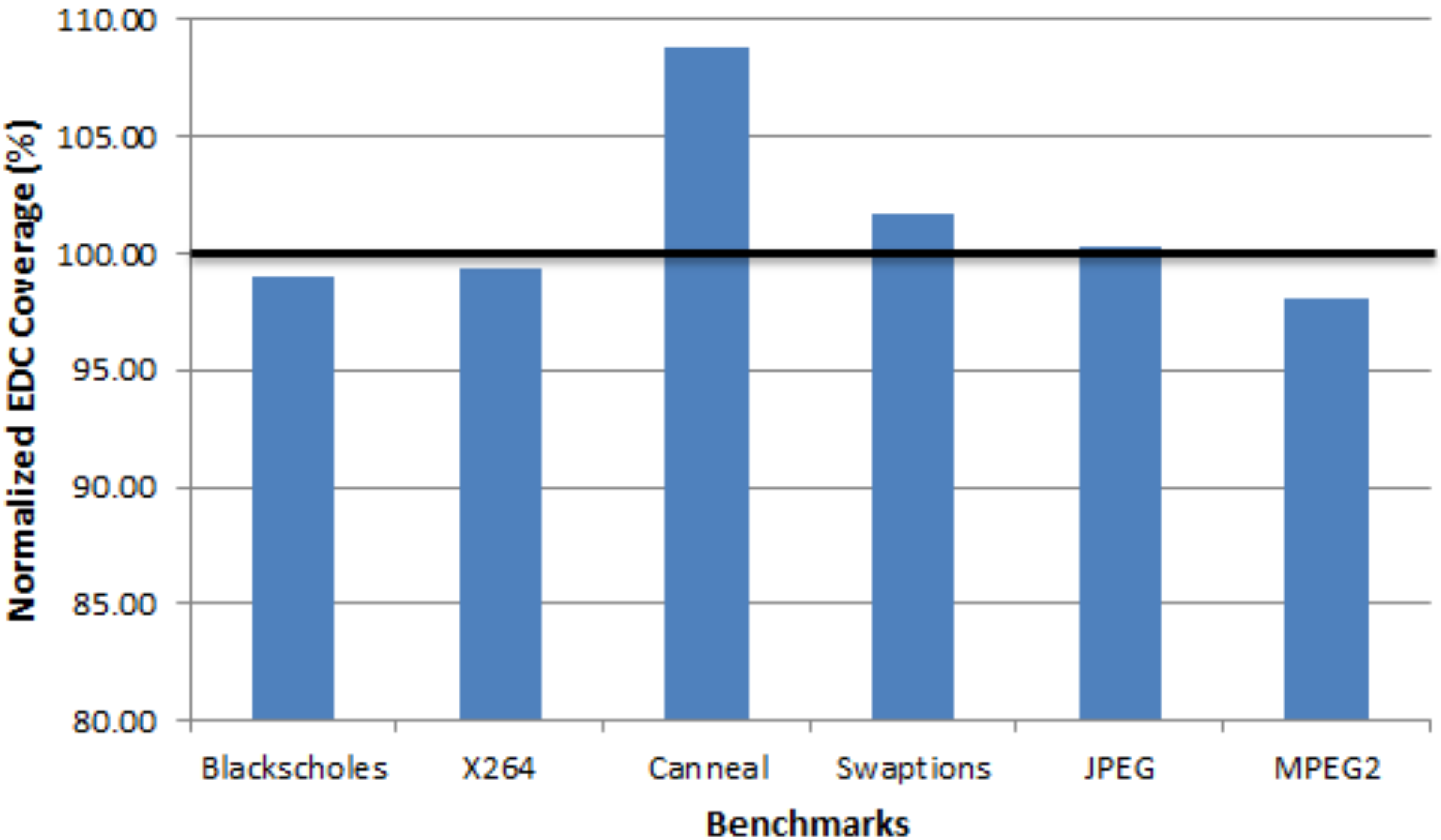
Baseline Resilience: EDC Percentages

Bench-mark	Inst-Combine	LICM	Loop-Unroll	SCCP	UnOpt
Blackscholes	9.9	9.48	8.48	9.38	10
X264	2.48	2.96	2.82	2.1	2.24
Canneal	4.56	3.26	3.26	3.94	3.32
Swaptions	2.5	3.12	2.44	2.98	2.36
JPEG	3.68	3.56	4.16	4.08	3.76
MPEG2	2	2.1	2.56	1.7	2.3
Average	4.19	4.08	3.95	4.03	3.99

EDC Coverage: InstCombine



EDC Coverage: LICM



Outline

- Background
- Problem and Approach
- Experimental Setup and Results
- Conclusion

Conclusion and Future Work

- Compiler optimizations: Characterize performance-resilience trade off in soft computing applications
  - Baseline resilience lowered in some benchmarks
  - Our technique preserves resilience in most cases

Future Work

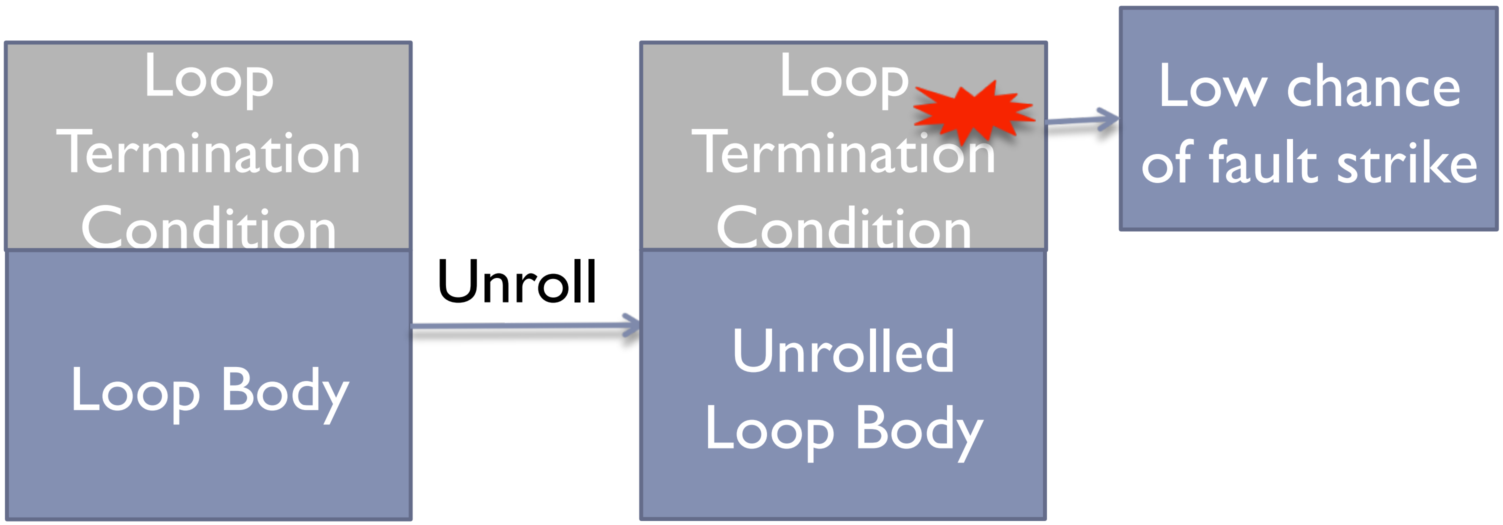
- Include more optimizations – ongoing work
- Classify optimizations into resilience packages
- Decision tree for applying optimizations

Contact: Anna Thomas <annat@ece.ubc.ca>

Backup Slides

Compiler Optimizations

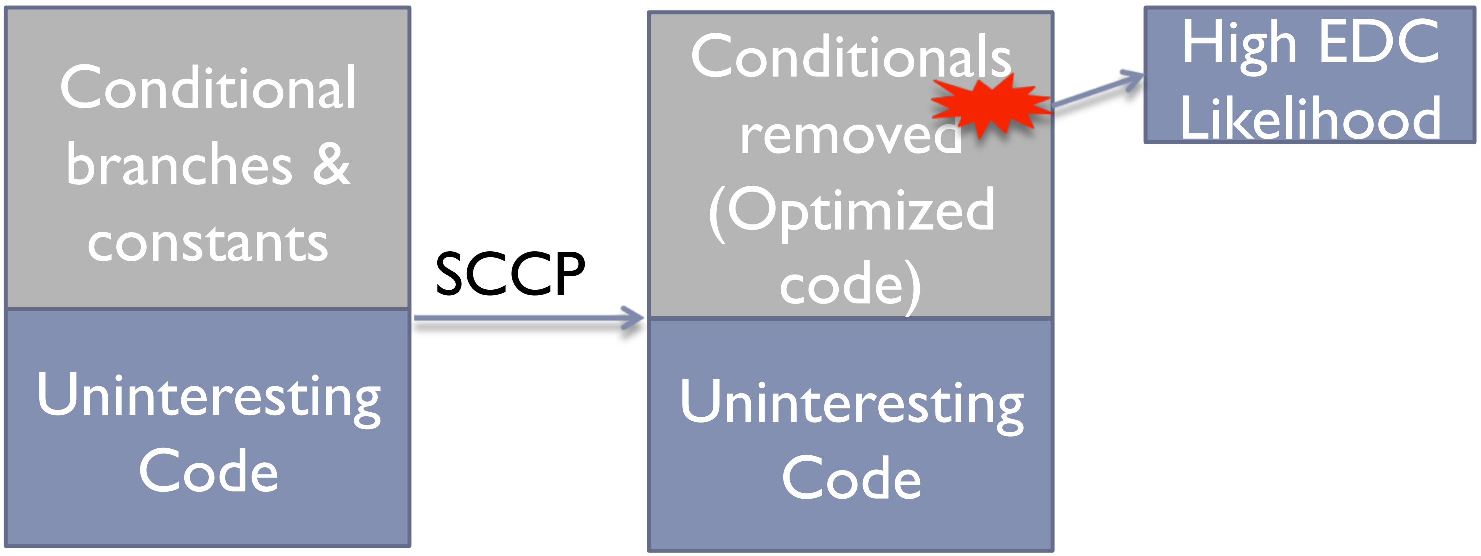
Loop Unrolling: Loop-Unroll



Lower EDC rate on Loop-Unrolled Code?

Compiler Optimizations

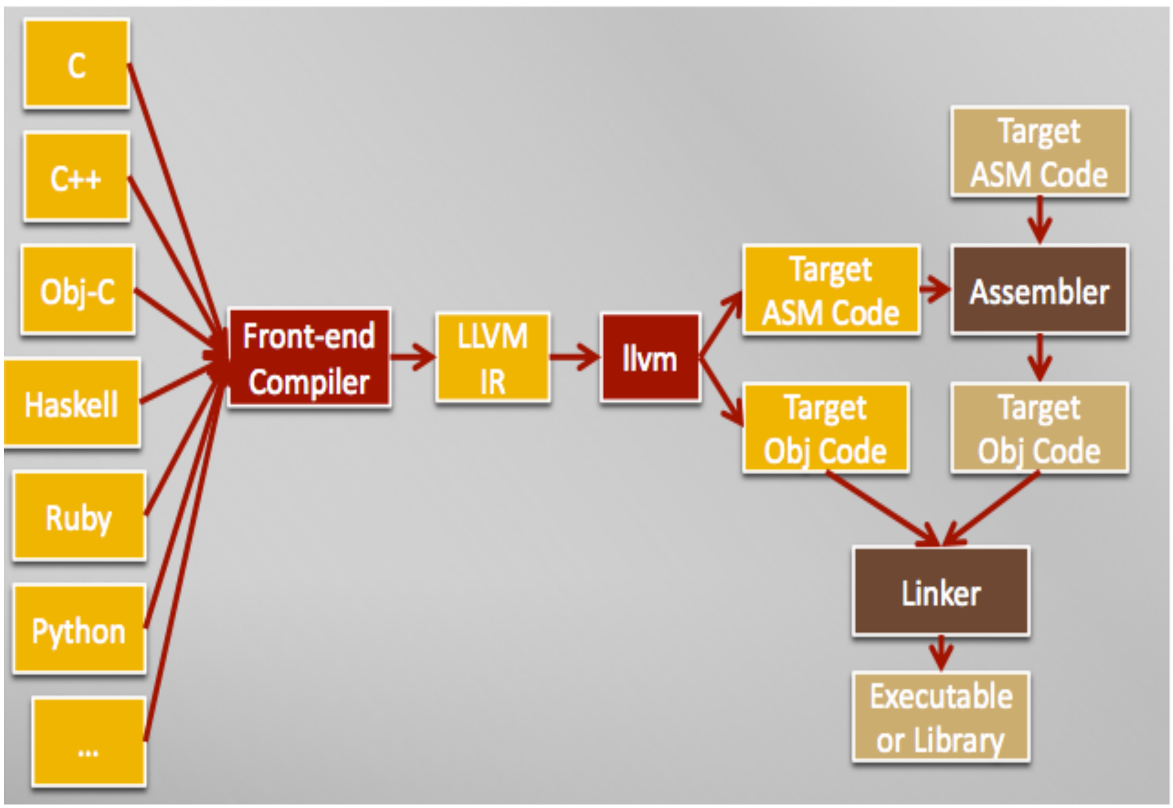
Sparse Conditional Constant Propagation: SCCP



Higher EDC rate on SCCP optimized Code?

LLVM Fault Injector LLFI

LLFI: Fault Injector at LLVM compiler's intermediate code level [Thomas'13]



- easy source code mapping