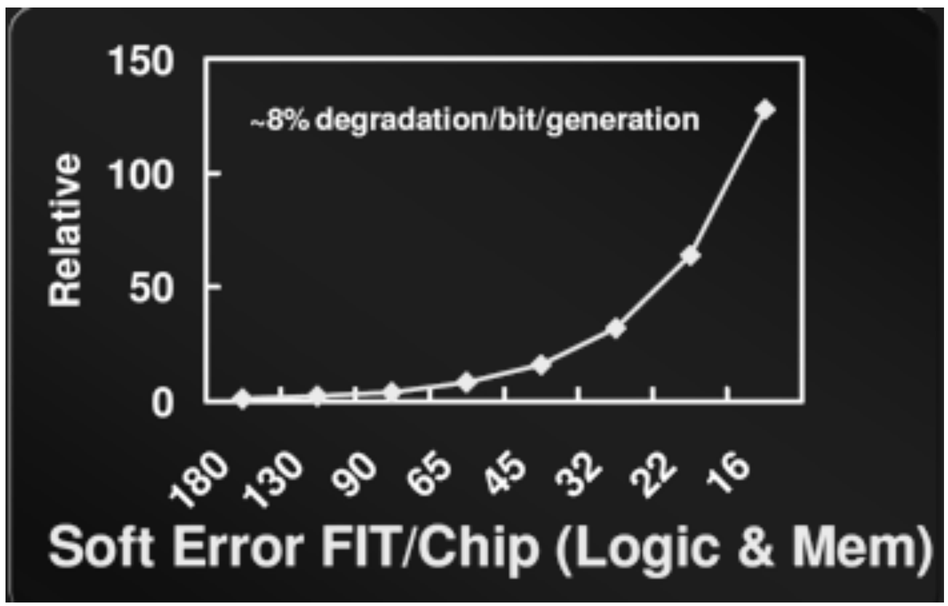
Finding Resilience-Friendly Compiler Optimizations using Meta-Heuristic Search Techniques

Nithya Narayanamurthy
Karthik Pattabiraman
Matei Ripeanu

University of British Columbia (UBC)

Motivation: Hardware Errors

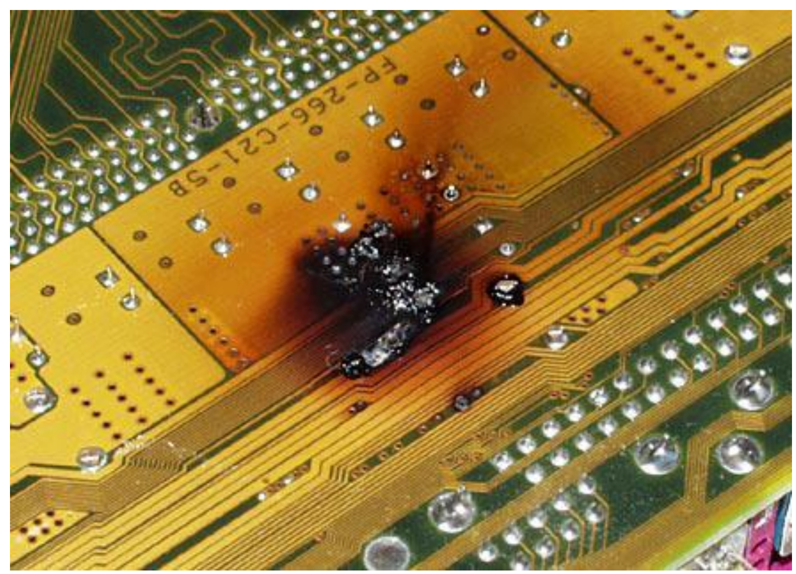
- Soft errors are becoming more common in processors
  - Caused by alpha particles and cosmic rays
  - More sensitive to errors as feature sizes shrink



Source: Shekar Borkar (Intel) - Stanford talk

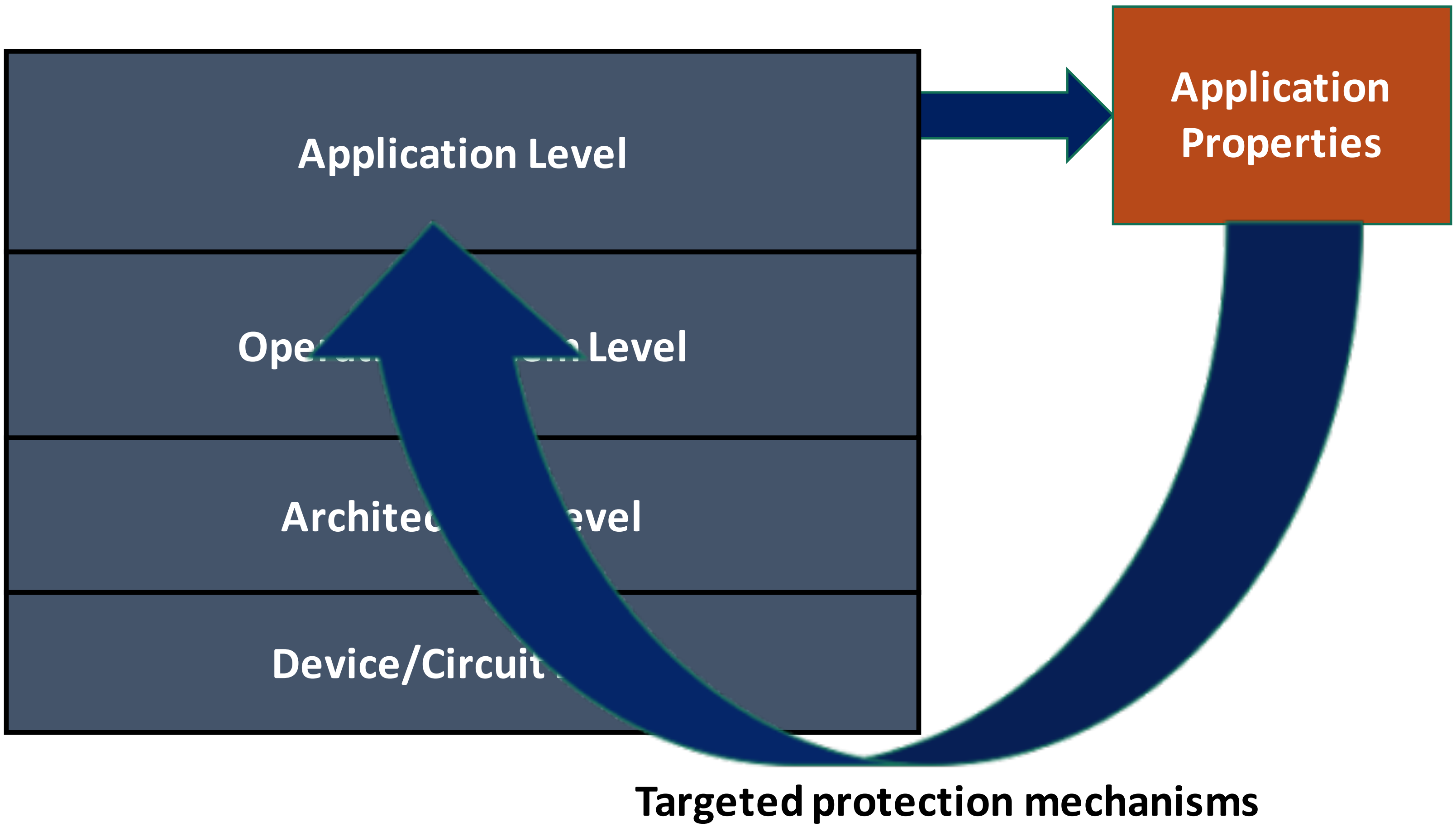
Traditional Solutions

- Dual Modular Redundancy (DMR) (e.g., IBM Mainframes, Tandem Non-Stop etc.)
- DMR incurs significant energy overheads - undesirable in commodity systems



Our Approach: Good Enough Dependability

Leverage the properties of the application to provide targeted protection, only for the errors that matter to it

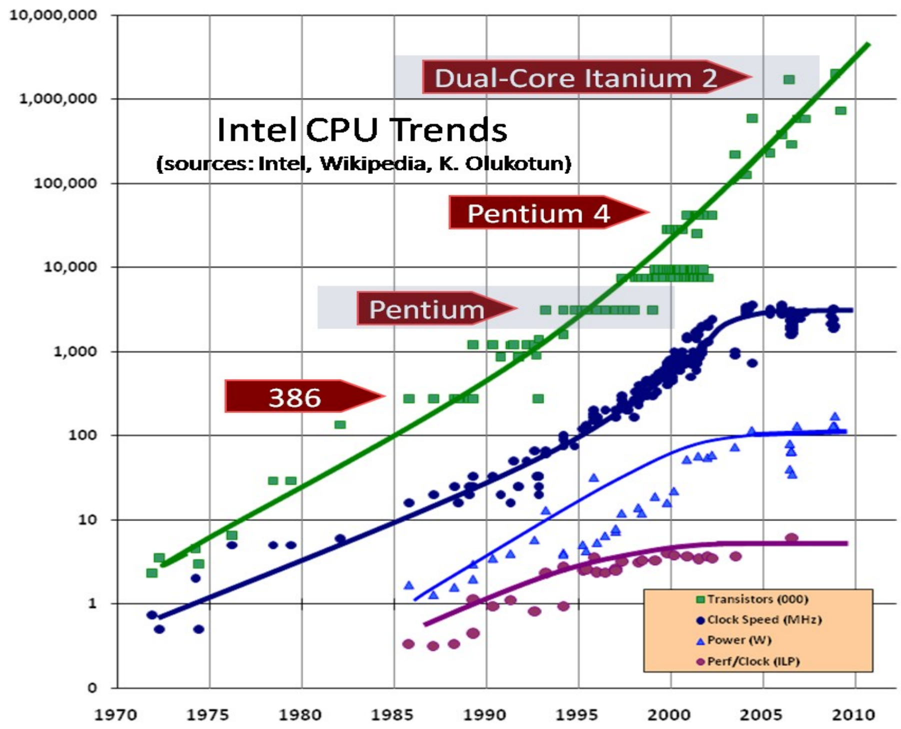


Error Resilience

- Silent Data Corruption (SDC)
  - Incorrect output without any indication (Most critical)
- Error Resilience:
  - Conditional probability that a fault does not produce an SDC given that it has occurred
- Vulnerability:
  - Product of error resilience and execution time of program
  - Assume transient faults occur uniformly over time

Compiler Optimizations

- Typically used to make programs faster by transforming the code
  - Processor performance has plateaued for a decade

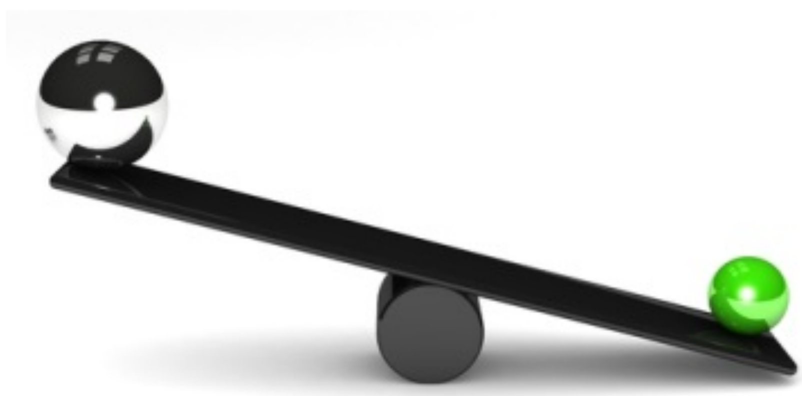


This Paper

Compiler Optimizations

Improve Performance! Resilience

Goal: Identify optimizations that **provide same** error resilience



Performance - Resilience trade off

RQ1: What effect do compiler optimizations have on a program's resilience?

Fault Injection Study

- **Fault Model:** Faults that occur in the computational components and register files of the processor

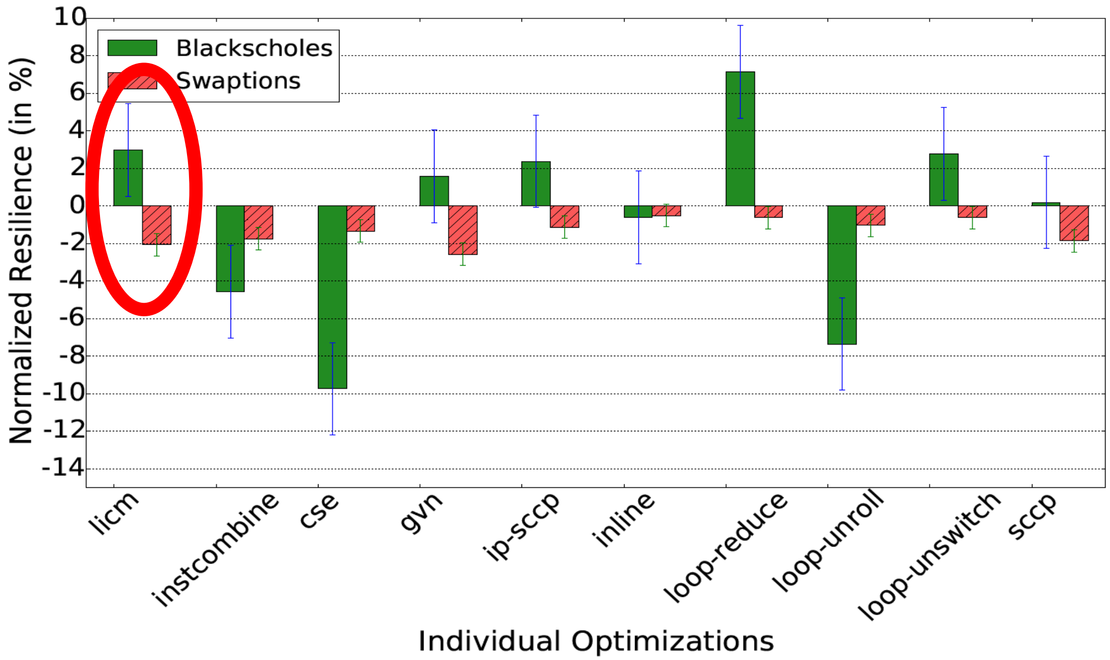
- Single bit flip fault

- One fault per run

- Injected using LLFI fault injector

Fault Injection Study

- 10 random compiler optimizations from LLVM compiler
- Benchmark programs – Blackscholes and Swaptions (Parsec)
- 3000 fault injections
- Error bars: 1.8%, 0.7%
- Resilience normalized to the resilience of unoptimized program



Same optimization can have different effects for different programs

Example optimization - 1: Loop Invariant Code Motion (LICM)

Un-optimized

```
for(i = 0; i < 10; i++)
{
    alpha = (x * c) + s;
    rs1[i] = i + (alpha * 7);
}
```

Optimized

```
alpha = (x * c) + s;
for(i = 0; i < 10; i++)
{
    rs1[i] = i + (alpha * 7);
}
```

LICM reduces overall resilience

Example Optimization - 2: LOOP-REDUCE

Un-optimized

```
alpha = (x * c) * s;
for(i = 0; i < 10; i++)
{
        rs1[i] = i * alpha;
}
```

Optimized

```
alpha = (x * c) * s;
temp = &rs1;
temp1 = 0;
for(i = 0; i < 10; i++)
{
  *temp = temp1 * alpha;
  temp1 = temp1 + 1;
  temp = temp+sizeof(int);
}
```

LOOP-REDUCE improves resilience

RQ1: Summary

- Different optimizations have different effects (degrade/improve/no impact) on error resilience
- Optimization's effect on resilience differs based on the application characteristics – **no one size fits all**

We need techniques for finding resilience-friendly optimization for a given application

RQ2: Can we find resilience-friendly optimizations that preserve the error resilience of a given program ?

Finding Resilience-Friendly Optimizations

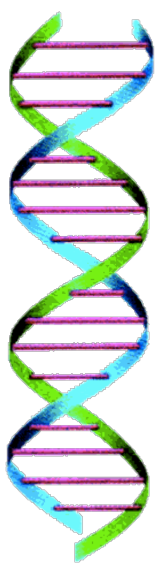
- Impact of an optimization depends on:
  - Application
  - Hardware platform
- Error resilience is sensitive to the order of optimizations in a sequence
- Search space is very large (2 ^ n * n!), where n is the number of optimizations

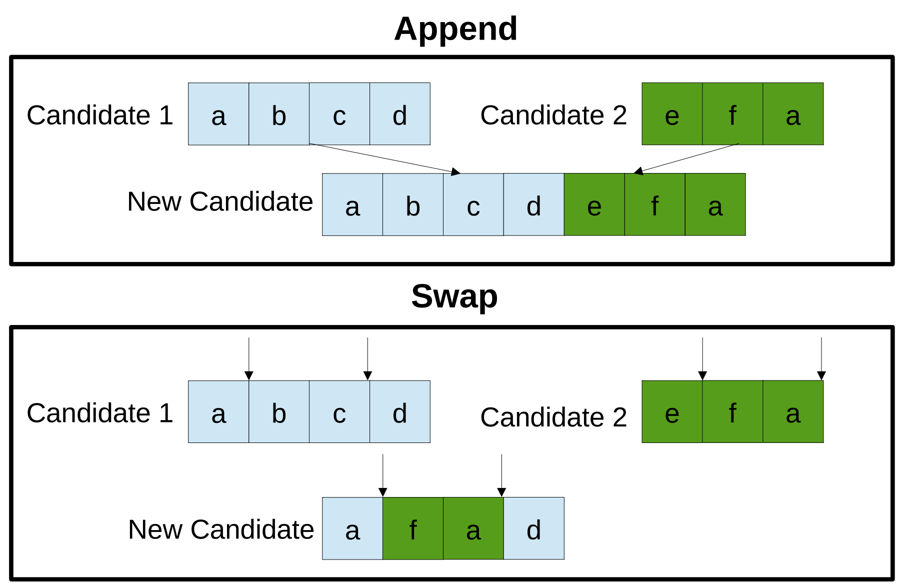
Meta-Heuristic Search Techniques

Genetic Algorithm (GA)

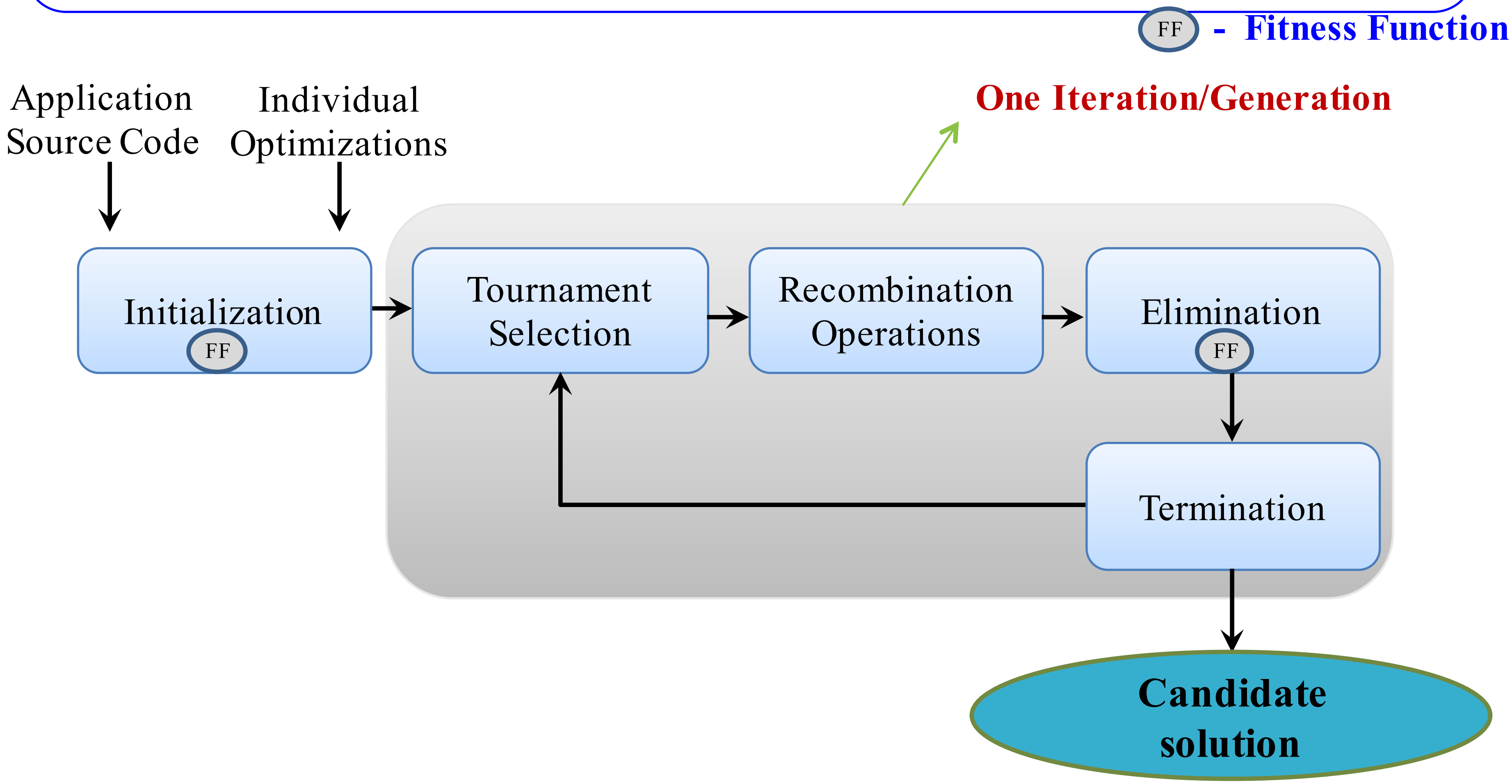
Evolve the solution by mutating it in every iteration till we converge to a candidate solution

- Kill the weak mutants based on fitness function

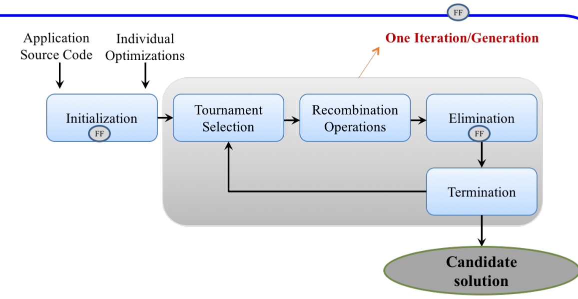




GA-based Approach



GA-based Approach



Initialization

instcombine

licm

Loop-unroll

gvn

inline

sccp

Loop-unswitch

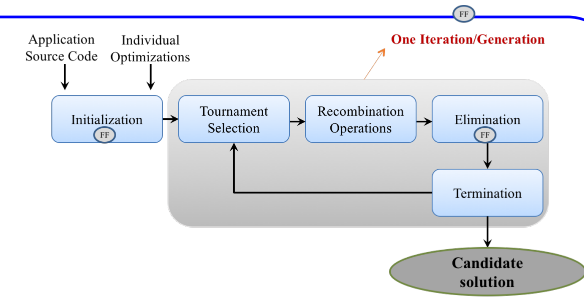
cse

Loop-reduce

Population

Optimization	Fitness Score
	57
	59
	50
	59
	57
	61
	64
	52
	55

GA-based Approach



Recombination Operations

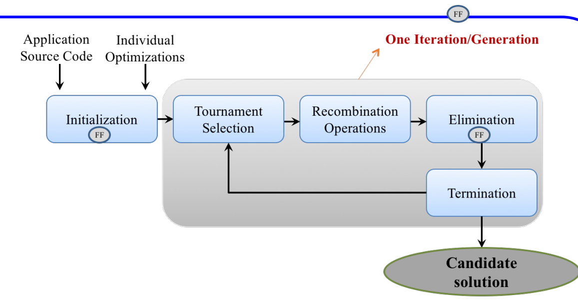
Tournament Selection

Optimization	Fitness Score
licm	57
instcombine	59
cse	50
inline	59
gvn	57
Loop-unswitch	61
Loop-unroll	64
sccp	52
Loop-reduce	55

Crossover - Append

-loop-reduce -loop-unroll	60

GA-based Approach



Recombination Operations

Optimization	Fitness Score
licm	57
instcombine	59
cse	50
inline	59
gvn	57
Loop-unswitch	61
Loop-unroll	64
sccp	52
Loop-reduce	55

Crossover - Append

-loop-reduce -loop-unroll	60

Experimental Setup

- **Twelve Benchmark programs**
  - Five from PARSEC
  - Seven from Parboil

- **LLVM compiler (widely used in industry)**

- **Performed fault injections using LLFI [DSN'14]**
  - 1000 fault injections, one fault per run
  - Error bars: 0.85% - 2.5%
  - Total fault injections: 60,000 fault injections

Evaluation

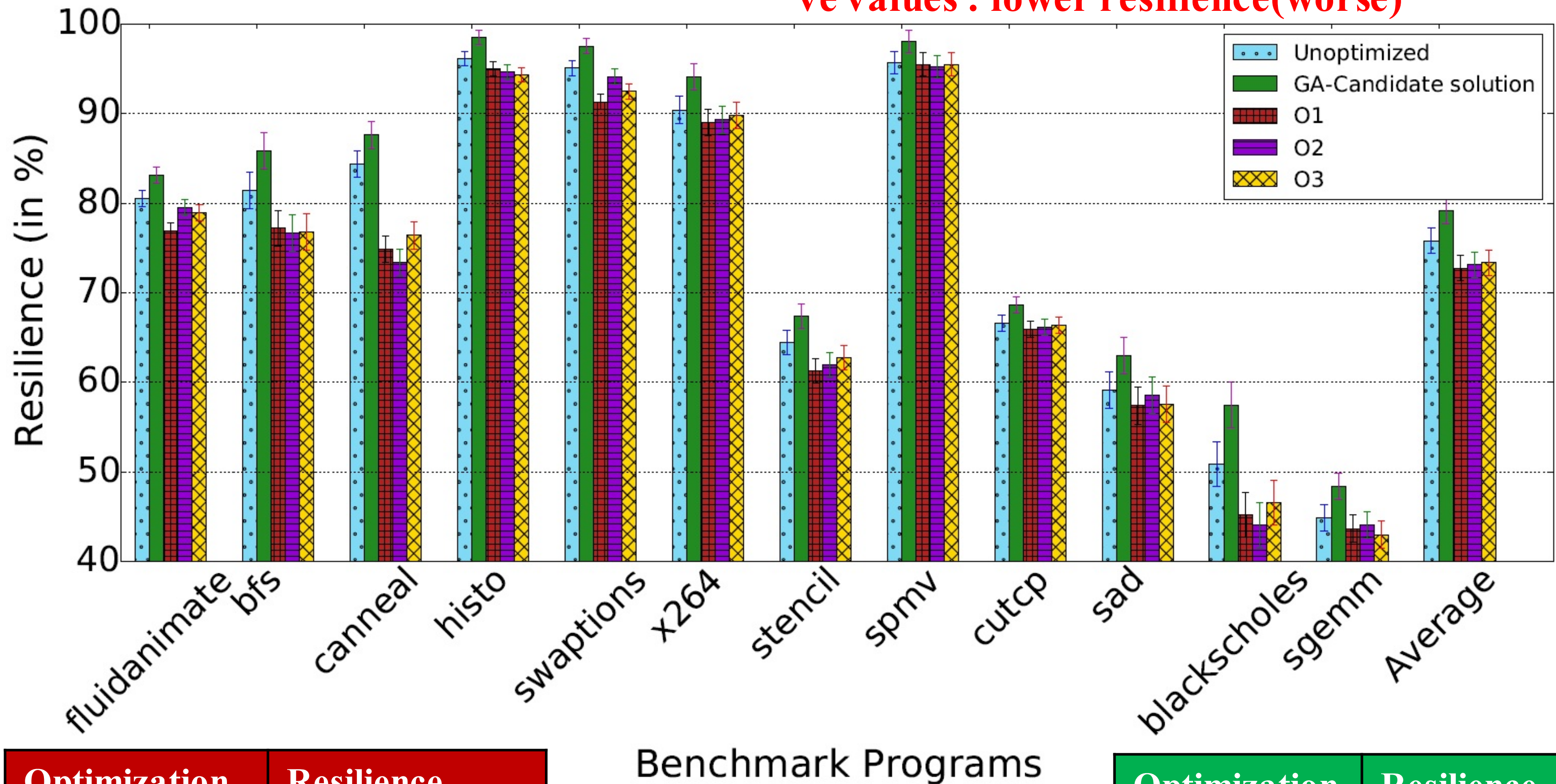
- Resilience
- Execution Time
- Vulnerability

Compared with standard levels O1, O2 and O3 as most prior work focuses on the standard levels [Sangchoolie'14][Demertzi'10]

Results: Resilience

+ ve values : higher resilience (better)

− ve values : lower resilience(worse)



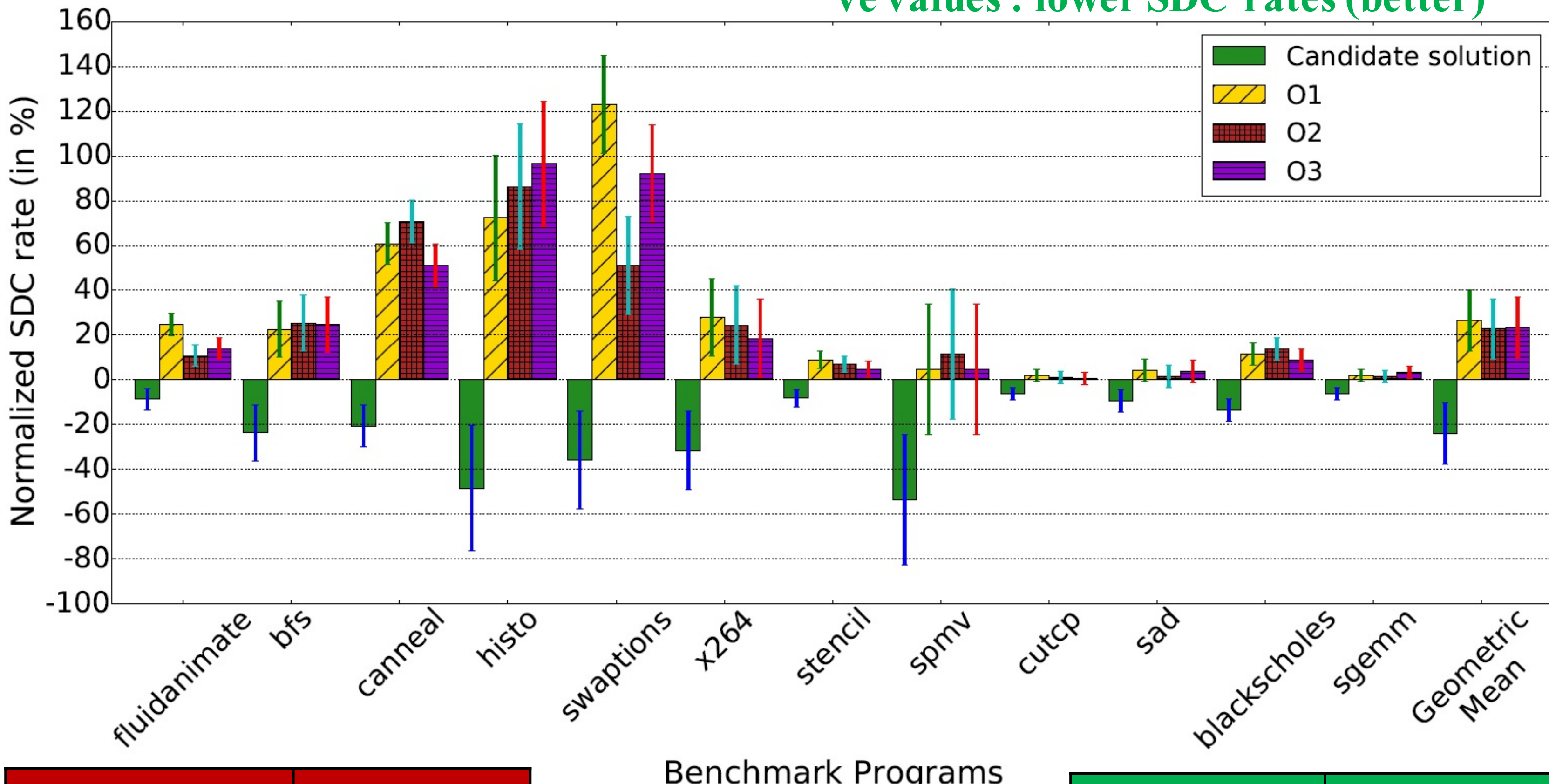
Optimization	Resilience Reduction
O1	-4.64%
O2	-4.21%
O3	-3.80%

Optimization	Resilience Improvement
GA	+4.46%

Results: Normalized SDC Rates

+ ve values : higher SDC rate (worse)

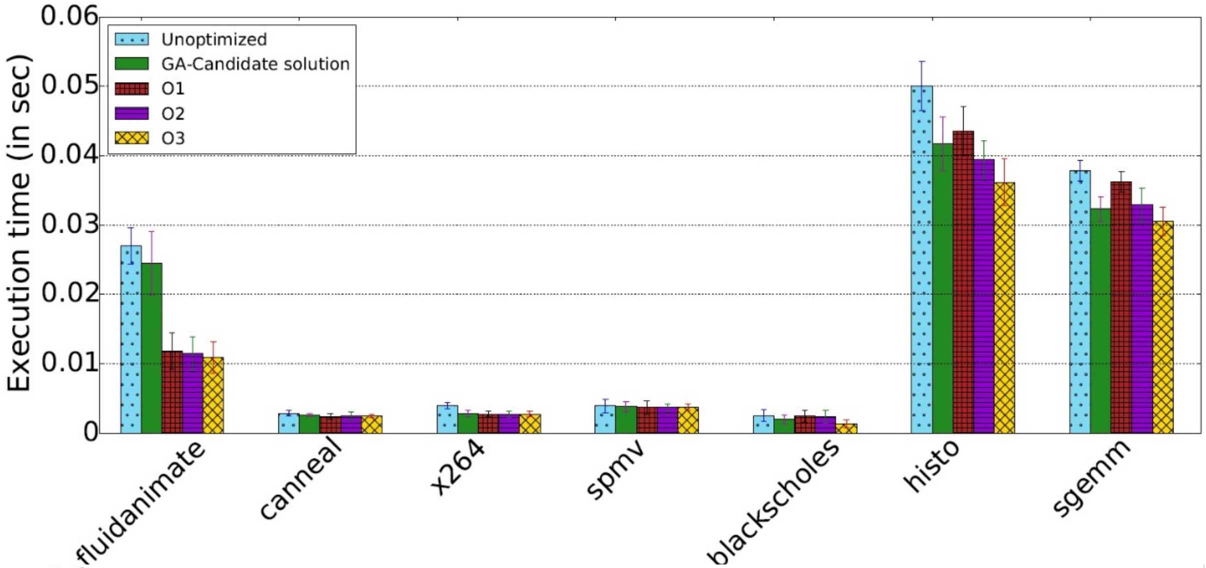
− ve values : lower SDC rates (better)

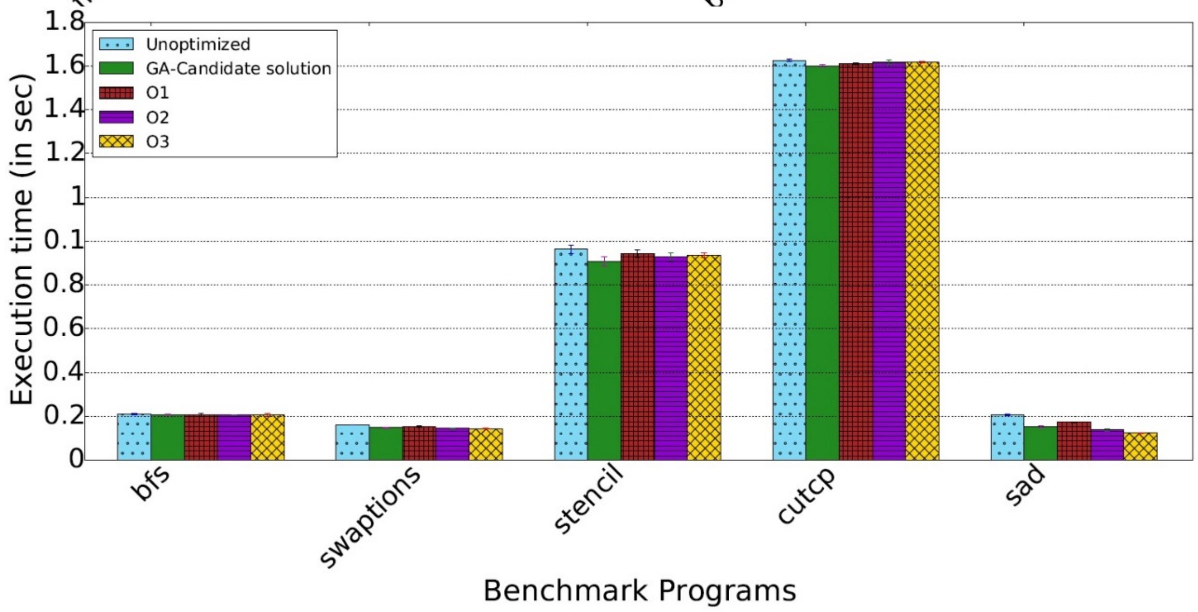


Optimization	SDC rate Increase
O1	+26.50%
O2	+22.79%
O3	+23.38%

Optimization	SDC rate Reduction
GA	-23.99%

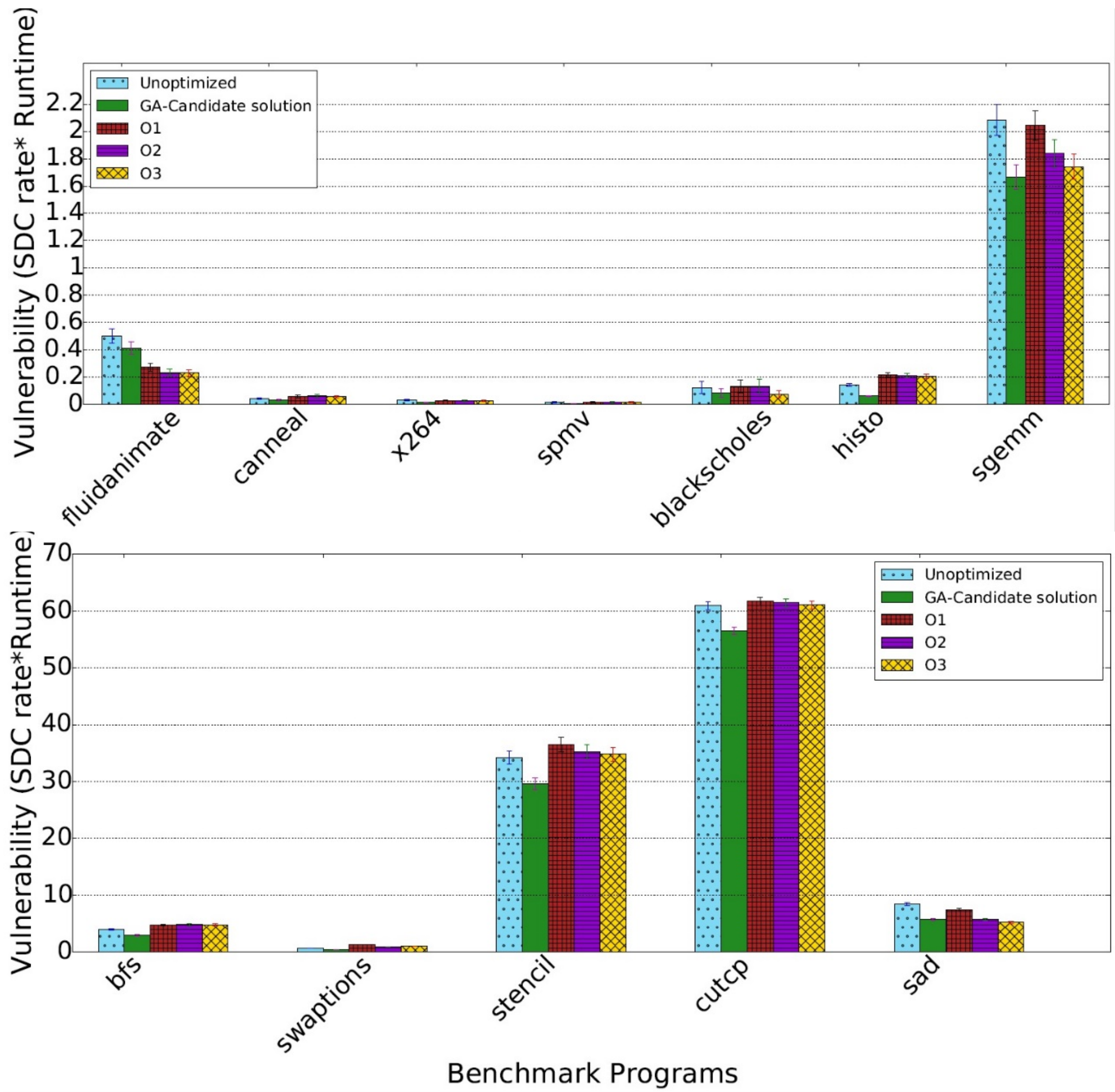
Results: Execution Time





Optimization	Performance Improvement
O1	6.56%
O2	7.21%
O3	11.10%
GA	8.99%

Results: Vulnerability



Optimization	Vulnerability Reduction
GA	-33.46%
O3	-6.26

Optimization	Vulnerability Increase
O1	+7%
O2	+0.35%

Vulnerability = SDC rate*run time

RQ2: Summary

- **Candidate solutions using Genetic Algorithms**
  - Outperform optimization levels in resilience
  - Provide reasonable performance improvements better than O1, O2 and only slightly worse than O3 (by about 2%)
- **Significant vulnerability reduction (by 27%) compared to O3, and much better than O1 and O2**
- **Takeaway: It is possible to achieve both high performance and high resilience using optimizations**
  - But they must be carefully chosen on a per-application basis

Related Work

- Analyzed the impact of standard levels O1, O2 and O3 on program's resilience[1 - Demertzi][2 - Sangchoolie]
- Proposed new optimizations that targets vulnerability reduction, without performance improvement [3 - Rehman]
- Focused on soft computing applications for EDC (Egregious Data Corruptions) outcomes [4 - Thomas][5 - Cong]

[1] Demertzi, Melina, Murali Annavaram, and Mary Hall. "Analyzing the effects of compiler optimizations on application reliability." *Workload Characterization (IISWC), 2011 IEEE International Symposium on*. IEEE, 2011.
[2] Sangchoolie, Behrooz, et al. "A Study of the Impact of Bit-Flip Errors on Programs Compiled with Different Optimization Levels." European *Dependable Computing Conference (EDCC), 2014 Tenth European*. IEEE, 2014.
[3] Rehman, Semeen, et al. "Reliable software for unreliable hardware: embedded code generation aiming at reliability." *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2011.
[4] Thomas, Anna, Jacques Clapauch, and Karthik Pattabiraman. "Effect of compiler optimizations on the error resilience of soft computing applications." In AER (2013).
[5] J. Cong and C. H. Yu. Impact of loop transformations on software reliability.In ICCAD, 2015

Conclusion

- **Studied the impact of individual compiler optimizations on error resilience**
    - Observed varied effects of optimizations on different programs
- **Used Genetic Algorithms (GAs) to find resilience friendly compiler optimizations for each program**
- **Candidate solutions have much better resilience and lower vulnerability than standard optimizations levels with only small performance degradation**