Error Propagation Analysis for Multi-Threaded Programs

Habib Saissi, Stefan Winter, Oliver Schwahn,
Karthik Pattabiraman, Neeraj Suri

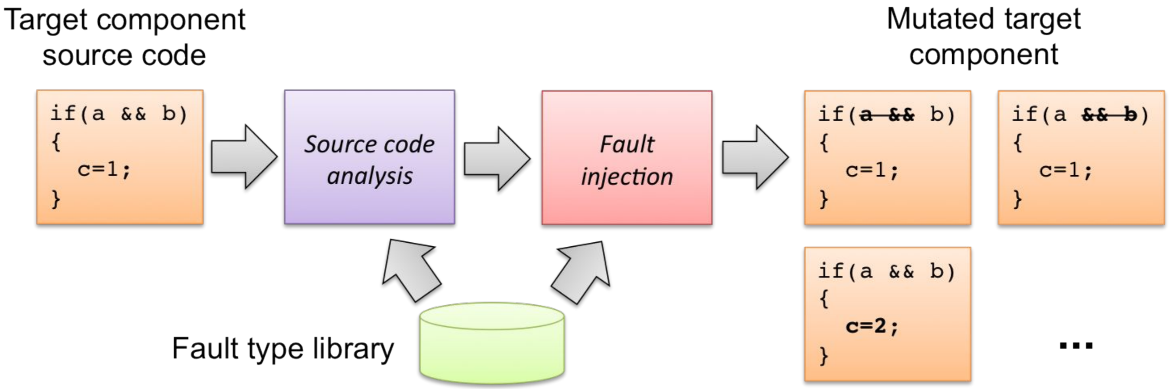
TECHNISCHE UNIVERSITÄT DARMSTADT

UBC

Lancaster University

Fault Injection

Evaluate the robustness of software



```
if(a && b)
{
  c=1;
}
```

```
if(~~a &&~~ b)
{
  c=1;
}
```

```
if(a ~~&& b~~)
{
  c=1;
}
```

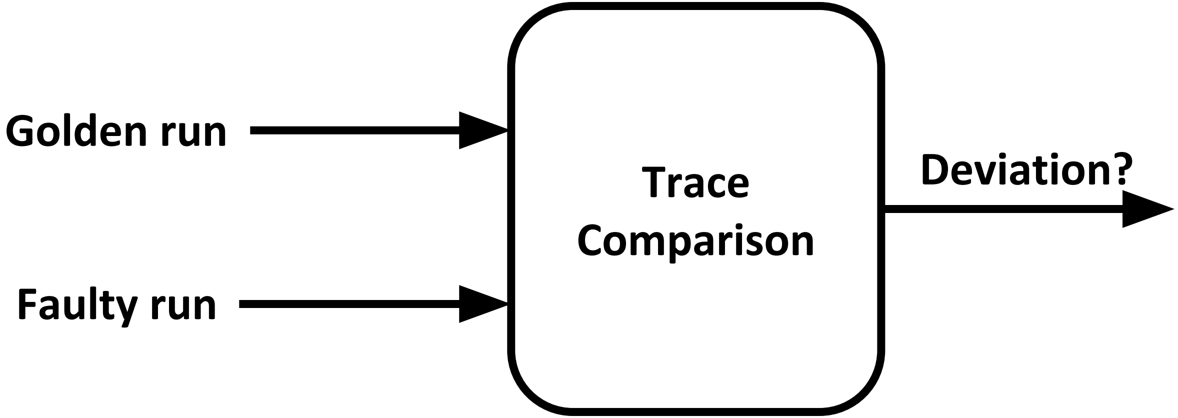
```
if(a && b)
{
  c=2;
}
```

Motivation: Error Propagation Analysis (EPA)

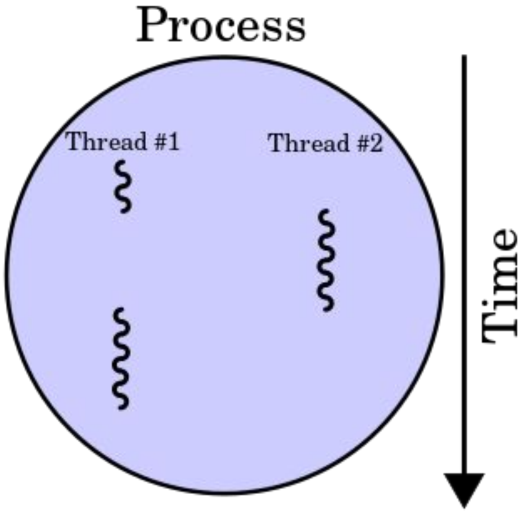
Compare FI run with golden run (fault free run)

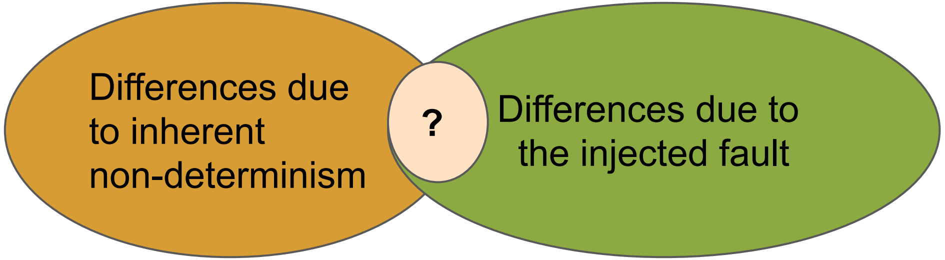
Any deviation indicates error propagation

Error Propagation Analysis



What about Multi-threaded programs ?





Differences due to inherent non-determinism

?

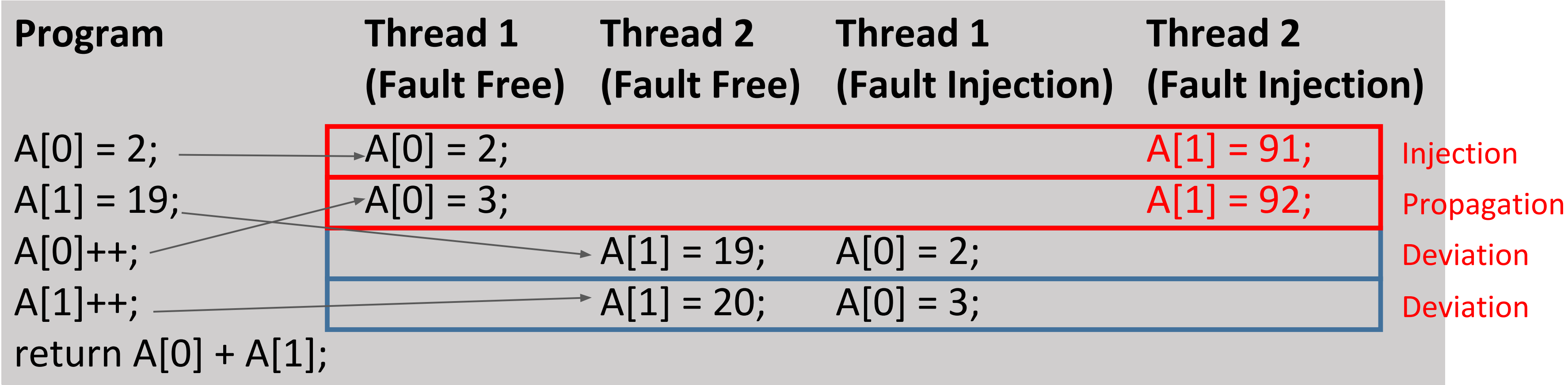
Differences due to the injected fault

Is the difference due to the non-determinism of multi-threading OR error propagation ?

Example: Single-Threaded EPA

Program	Fault-free Run	Fault Injection	
A[0] = 2;	A[0] = 2;	A[0] = 2;	
A[1] = 19;	A[1] = 19;	A[1] = 91;	Injection
A[0]++;	A[0] = 3;	A[0] = 3;	
A[1]++;	A[1] = 20;	A[1] = 92;	Propagation
return A[0] + A[1];	return 23;	return 94;	Propagation

Example: Multi-threaded EPA



Program	Thread 1 (Fault Free)	Thread 2 (Fault Free)	Thread 1 (Fault Injection)	Thread 2 (Fault Injection)	
A[0] = 2;	A[0] = 2;			A[1] = 91;	Injection
A[1] = 19;	A[0] = 3;			A[1] = 92;	Propagation
A[0]++;		A[1] = 19;	A[0] = 2;		Deviation
A[1]++;		A[1] = 20;	A[0] = 3;		Deviation
return A[0] + A[1];					

Our Work: TraceSanitizer

First **sound** technique to disambiguate error propagation in **multi-threaded** programs from non-determinism (**without needing** any programmer **annotations)**

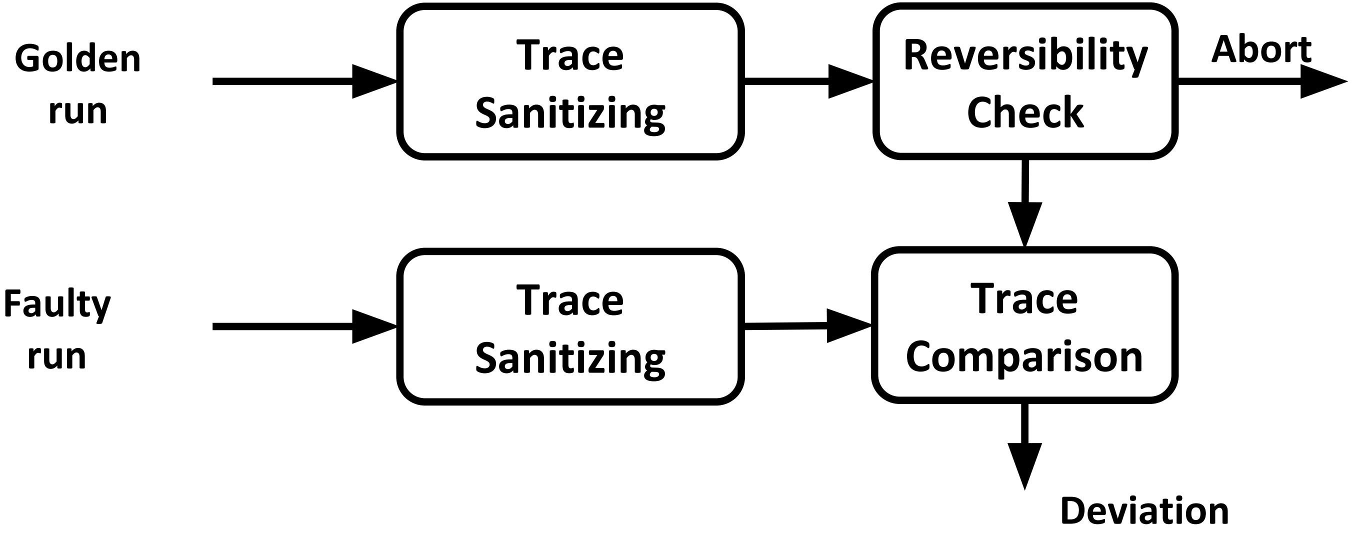
Intuition: Pseudo-deterministic condition

- An execution trace is pseudo-deterministic:
  - No **dependent** instructions that can occur in **reversed** order

- Pseudo-deterministic condition **guarantees** soundness

- Example: Map Reduce

TraceSanitizer: WorkFlow



Reversibility Check

Order constraints

$$\Phi = \psi \wedge R$$

Reversibility constraints

Example: TraceSanitizer Operation

Original Trace

```
0 call-pthread_create 0
   → 7ffcfe3282e8 0 400ae0 0
0 call-pthread_create 0
   → 7ffcfe3282e0 0 4012c0 0
1 call-inc 0
1 alloca 7f0ccbc55d58 8
1 alloca 7f0ccbc55d50 8
1 store 0 7f0ccbc55d50
2 call-inc 0
2 alloca 7f0ccb454d58 8
```

Sanitized Trace

```
T_0     call-pthread_create-u 0
        → o4 0 400ae0 0
T_0     call-pthread_create-u 0
        → o5 0 4012c0 0
T_0_0 call-inc 0
T_0_0 alloca o6 1 8
T_0_0 alloca o7 1 8
T_0_0 store 0 o7
T_0_1 call-inc 0
T_0_1 alloca o8 1 8
```

Evaluation

- Implemented as a pass in the LLVM compiler
- C/C++ programs from the PARSEC and Phoenix benchmarks
- Reversibility check with the Z3 SMT solver
- Injected 5 different types of software faults (5000 injections each)

False positives and Time Taken

Program	# Threads	False Positives	Reversibility Check Time
quicksort	72	**0**	**30 min**
pca	17	**0**	**150 min**
kmeans	65	**0**	**82 min**
blackscholes	3	**0**	**1 min**
swaptions	4	**0**	**145 min**

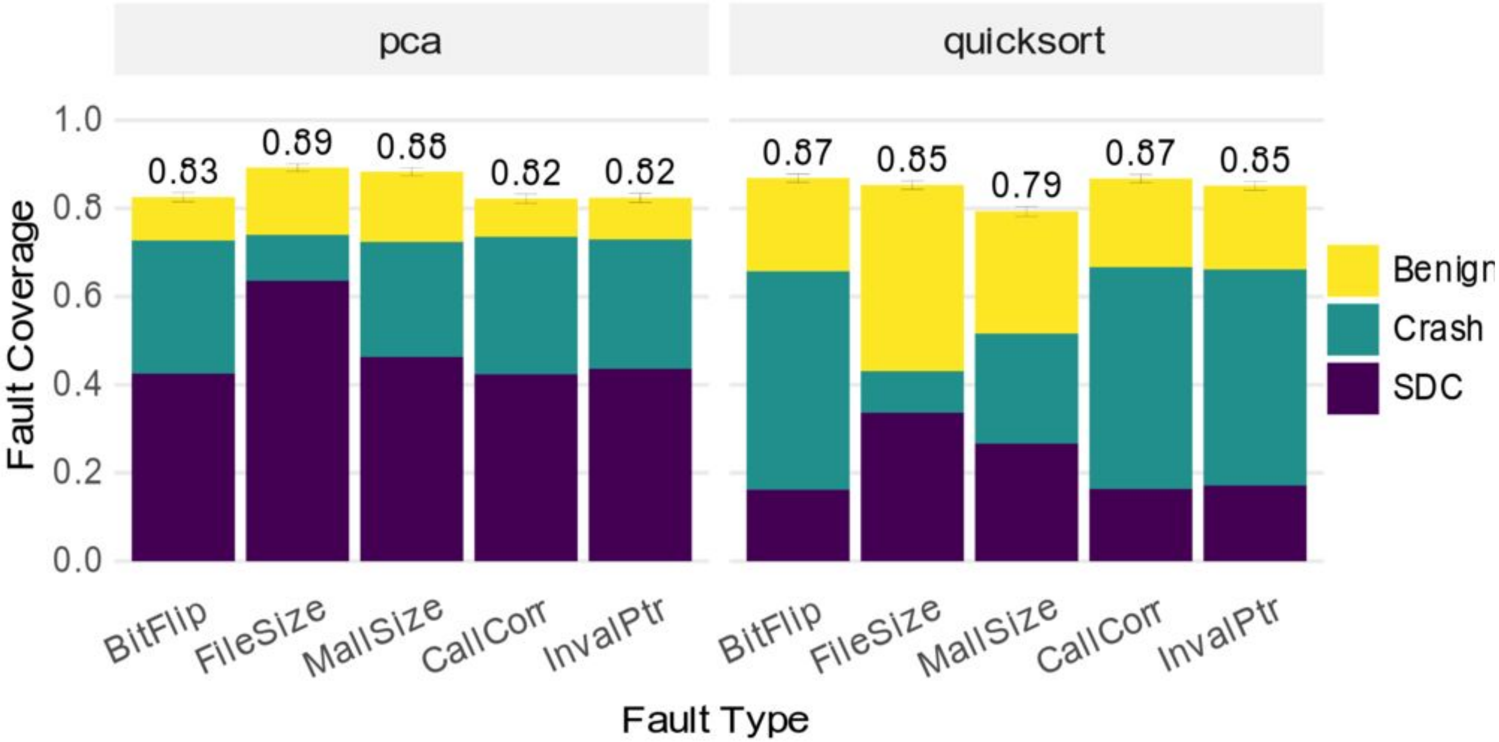
Fault Model

Residual software bugs that are hard to detect through regression or unit tests

Faults Considered:

- Bit Flip
- File I/O Buffer Overflow
- Buffer Overflow Malloc
- Function Call Corruption
- Invalid Pointer

Fault Injection Results



Summary

Non-Determinism in multi-threaded programs is bad for EPA

TraceSanitizer (TS): First **Sound** technique to perform EPA for a class of Multi-threaded programs (**pseudo-deterministic**)

- Condition encoded as reversibility check - SMT solvers
- Completely automated; no program annotations needed

Evaluation shows TS has **0% false-positives, incurs reasonable overheads and provides high fault coverage**

https://github.com/DEEDS-TUD/TraceSanitizer