IPA: Error Propagation Analysis of Multithreaded Programs Using Likely Invariants

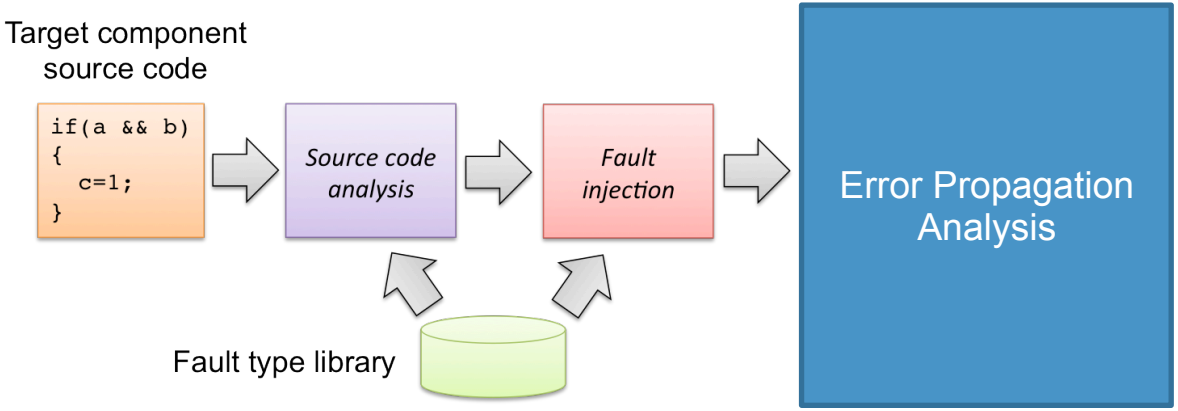
Abraham Chan*, Stefan Winter‡, Habib Saissi‡, Karthik Pattabiraman*, Neeraj Suri‡

*The University of British Columbia

‡Technische Universität Darmstadt

Fault Injection

- Evaluate the robustness of software
- Inject software simulated faults into programs



Error Propagation Analysis (EPA)

- Understand where and why errors manifest [Hiller et al.]
- Compare a golden trace with a faulty trace [Ammar et al.]
- What about multithreaded programs?

Our contribution:
EPA framework for multithreaded programs

EPA Example

```
1.  int func (int x) {
2.          int a = x - 3;
3.          if (a > 0) {
4.                  x = 1;
5.          } else {
6.                  x = 2;
7.          }
8.          a = a + 5;
9.          return x;
10. }
```

Trace for x = 4

1. x = 4, a = 1
2. x = 1, a = 1
3. x = 1, a = 6

Fault Detection using Tracing

```
1.  int func (int x) {
2.          int a = x - 3;
3.          if (a < 0) {    *Fault
4.                  x = 1;
5.          } else {
6.                  x = 2;
7.          }
8.          a = a + 5;
9.          return x;
10. }
```

Traces for x = 4

Faulty Trace	Golden Trace
1. x = 4, a = 1	1. x = 4, a = 1
2. x = 2, a = 1	2. x = 1, a = 1
3. x = 2, a = 6	3. x = 1, a = 6

Multithreading

```
1.  int func (int x) {
2.          int a = x - 3;
3.          if (a > 0) {
4.                  x = 1;
5.          } else {
6.                  x = 2;
7.          }
8.          a = a + 5;
9.          return x;
10. }
```

Traces for x = 4 on 2 threads

Thread 1	Thread 2
1. x = 4, a = 1	4. x = 4, a = 1
2. x = 1, a = 1	5. x = 1, a = 1
3. x = 1, a = 6	6. x = 1, a = 6
1, 2, 3, 4, 5, 6	1, 4, 5, 6, 2, 3

Who to trust?

Likely Invariants

- Statistically inferred from dynamic program traces with a confidence threshold
- Cheaper to infer than true invariants
- Likely Invariant Detection Tool: Daikon [Ernst et al.]
- Example: this.theArray.length >= 5

Invariant Example

Daikon Invariants inferred at x = 4

```
int func (int x) {
        int a = x - 3;
        if (a > 0) {
                x = 1;
        } else {
                x = 2;
        }
        a = a + 5;
        return x;
}
```

x == 4

x == 1, a > 0

Invariant Fault Detection

Daikon Invariants inferred at x = 4

```
1.  int func (int x) {
2.          int a = x - 3;
3.          if (a < 0) {    *Fault*
4.                  x = 1;
5.          } else {
6.                  x = 2;
7.          }
8.          a = a + 5;
9.          return x;
10. }
```

x == 4

x == 1, a > 0

Faulty Trace

1. x = 4, a = 1
2. x = 2, a = 1
3. x = 2, a = 6

Error Propagation

```
int func (int x) {
        int a = x - 3;
        if (a < 0) {    *Fault*
                x = 1;
        } else {
                x = 2;
        }
        a = a + 5;
        return x;
}
```

Invariants inferred at x = 4

x == 4

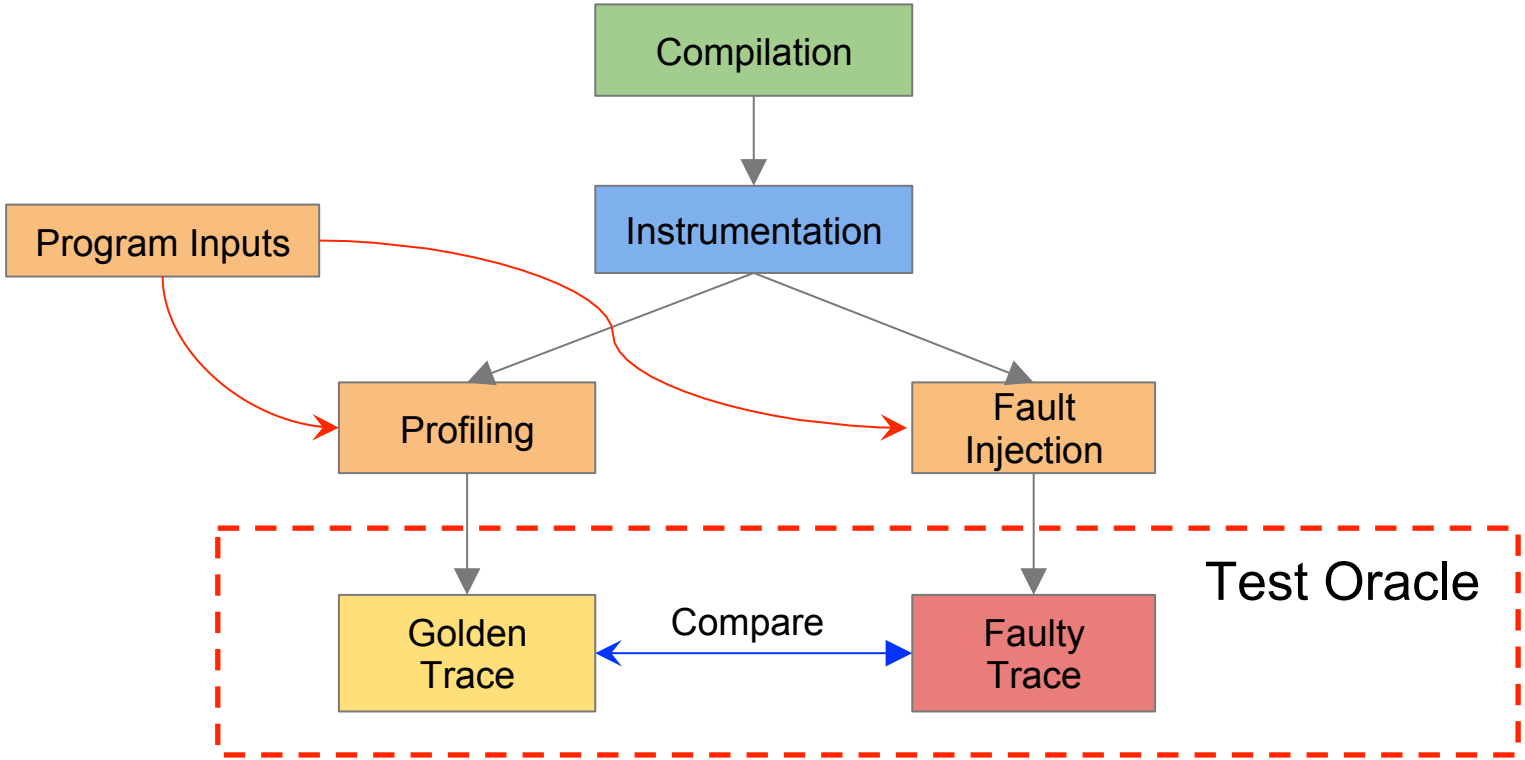
Fault has propagated through the function.

x == 1, a > 0

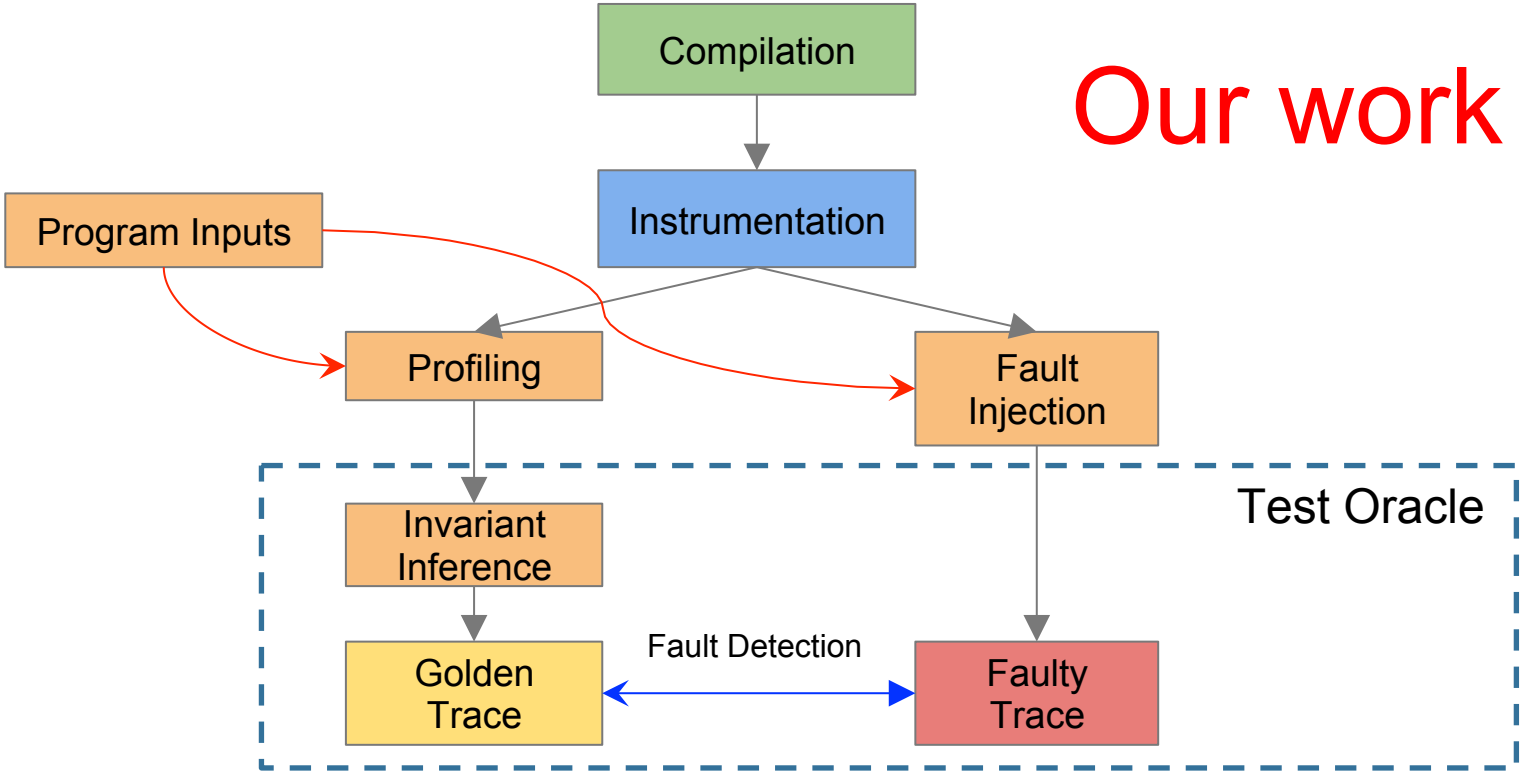
Main Contributions

- **Develop an EPA framework for multithreaded programs using likely invariants**
- Empirically assess the efficacy of invariants for fault detection

Traditional EPA Workflow



Invariant Propagation Analysis (IPA)



Main Contributions

- Develop an EPA framework for multithreaded programs using likely invariants
- **Empirically assess the efficacy of invariants for fault detection**

Research Questions

Q1: Stability
Q2: Fault Coverage
Q3: Performance

Experimental Setup

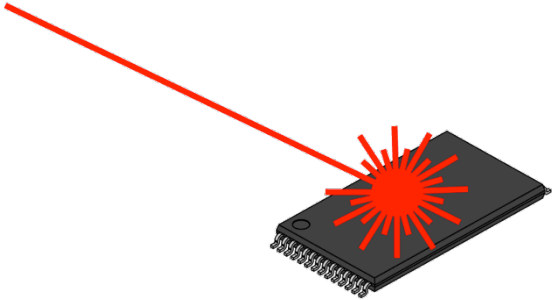
- 6 multi-threaded benchmarks
  - Domains: Sorting, scientific computing, web server
  - 3 from PARSEC suite

- Fault Injection Tool: LLFI
  - LLVM based fault injector
  - Developed at UBC's Dependable Systems Lab
  - https://github.com/DependableSystemsLab/LLFI

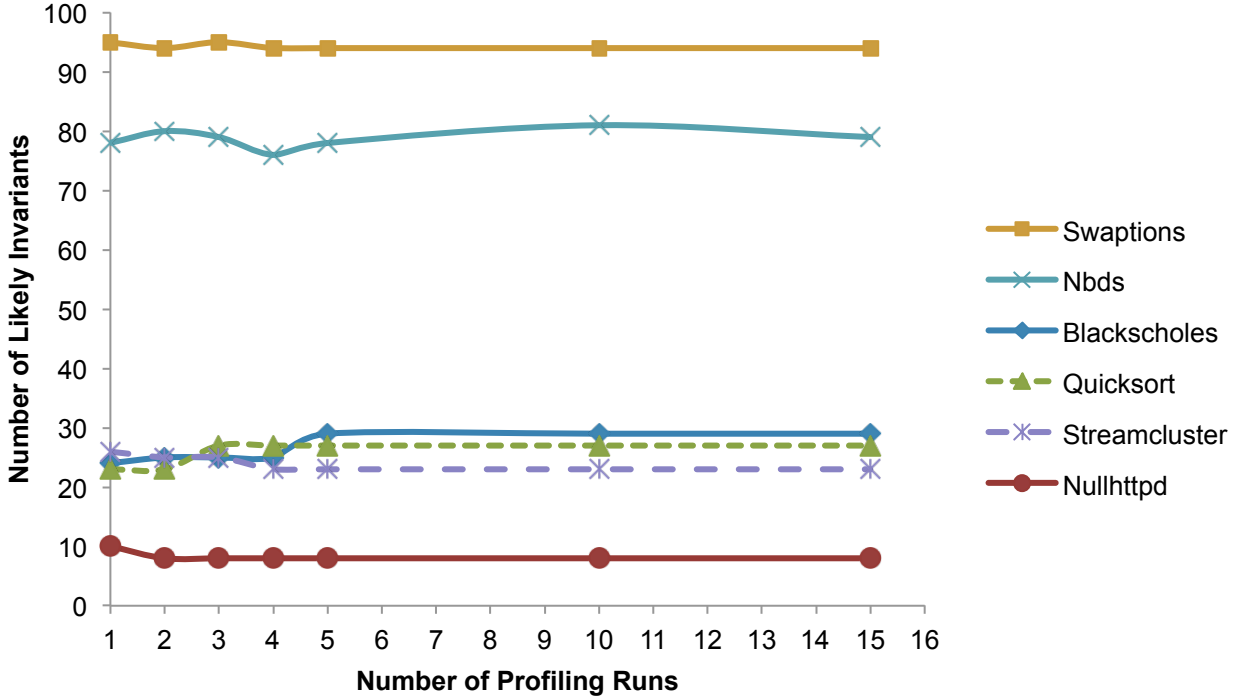
Fault Model

- Common software bugs that are hard to detect through regression or unit tests [Vipindeep et al.]

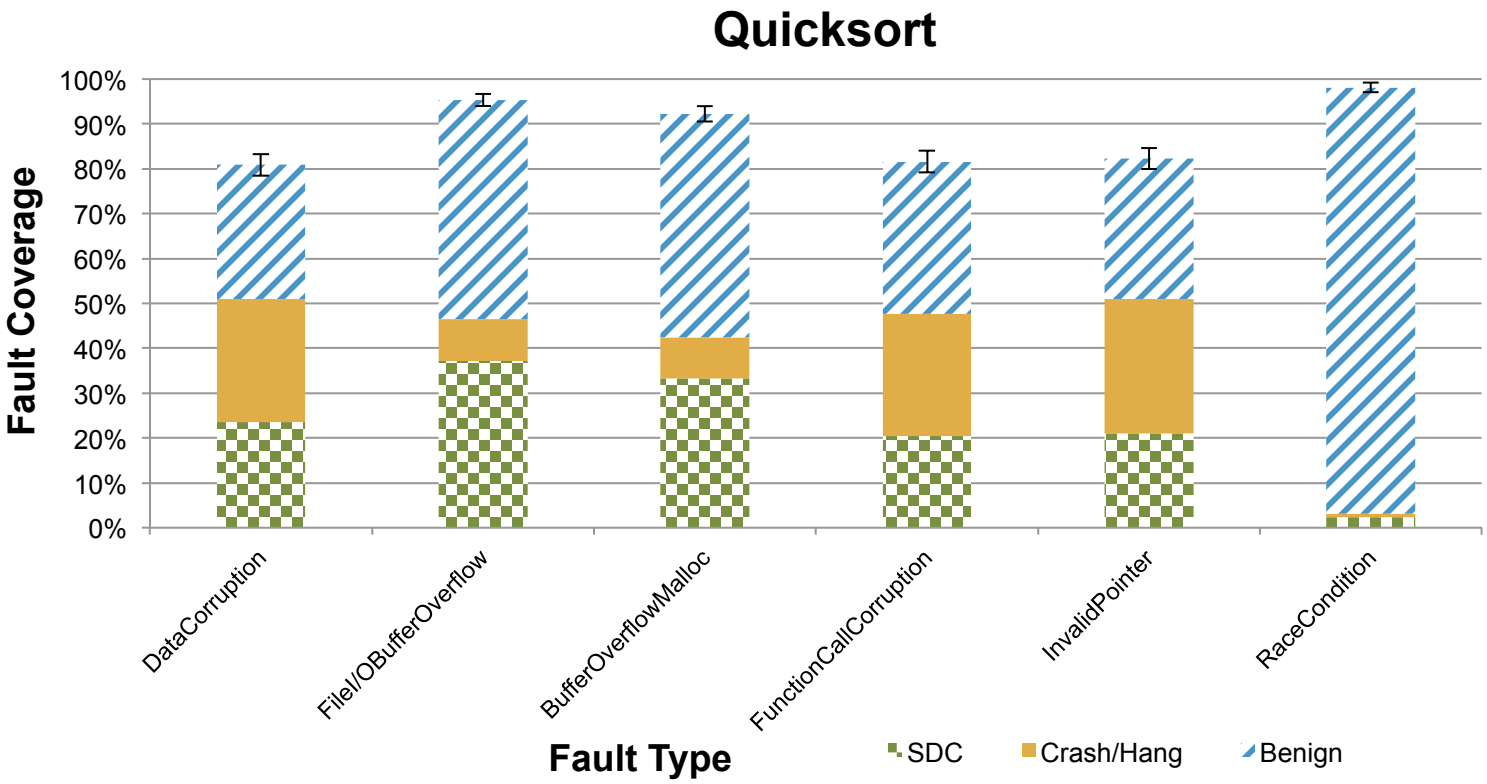
- Faults Considered:
  - Data Corruption
  - File I/O Buffer Overflow
  - Buffer Overflow Malloc
  - Function Call Corruption
  - Invalid Pointer
  - Race Condition



RQ1: Stability

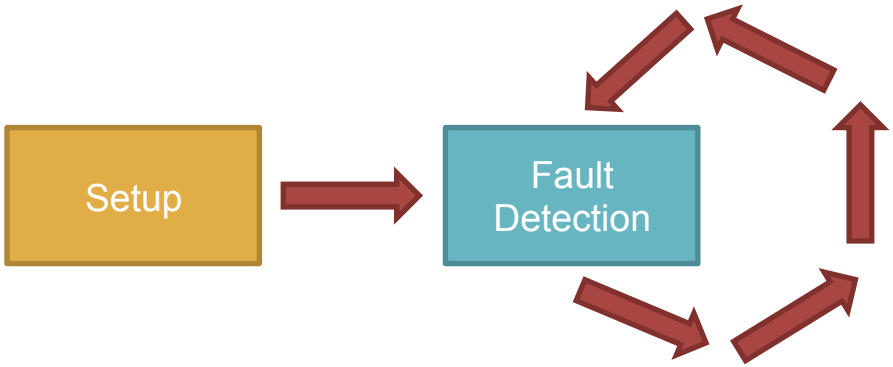


RQ2: Fault Coverage



RQ3: Performance

- **Setup Overhead:** IPA is 2-90% slower than EPA
- **Fault Detection:** IPA is 2.7x to 151x faster than EPA
- Fault Detection time is amortized over experimental runs



Lessons Learned

- Fault coverage is dependant on the application and fault type
- Possible trade off between invariant stability and fault coverage
- Certain types of invariants may be better at detecting different faults

Summary

- **Problem:** Multithreaded programs produce nondeterministic golden traces
- **Approach:** Use likely invariants to detect faults
- **Result:** Likely invariants offer good fault detection in many applications

- **Available at:** http://github.com/DependableSystemsLab/LLFI-IPA
- **Contact:** http://ece.ubc.ca/~abrahamc/