Determinism Is Not Enough: What Really Makes Multithreaded Programs Hard to Get Right and What Can Be Done about It?

Junfeng Yang, Heming Cui, Jingyue Wu,
Columbia University

One-slide summary

- Multithreaded programs: critical but hard to get right

- Many blamed *nondeterminism*
- Deterministic multithreading (DMT): one input → one schedule
- But, determinism is neither sufficient nor necessary for reliability!

- True culprit is rather quantitative: too many schedules
- Stable Multithreading (StableMT): all inputs → a small set of schedules

Background and motivation

Multithreaded programs: hard to get right

- Plagued with concurrency bugs
  - Data races, atomicity violations, order violations, deadlocks, etc
- Concurrency bugs: bad
  - Have taken lives in the Therac 25 incidents and caused the 2003 Northeast blackout
  - May be exploited by attackers to violate confidentiality, integrity, and availability of critical systems [Hotpar 12]

Concurrency bug examples

```
Thread 0                    Thread 1
mutex_lock(M)
*obj = …
mutex_unlock(M)
                            mutex_lock(M)
                            free(obj)
                            mutex_unlock(M)
```

```
Thread 0                    Thread 1
                            mutex_lock(M)
                            free(obj)
                            mutex_unlock(M)
mutex_lock(M)
*obj = …
mutex_unlock(M)
```

Apache Bug #21287

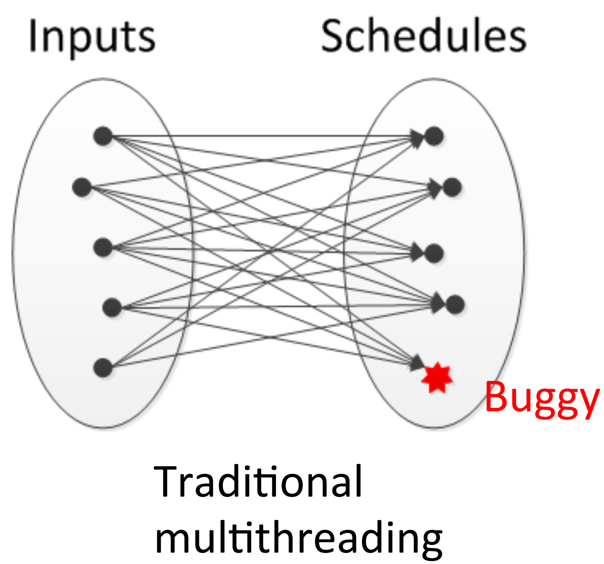
```
Thread 0                    Thread 1
……                          ……
barrier_wait(B)             barrier_wait(B)
                            result += …
print(result)
```

```
Thread 0                    Thread 1
……                          ……
barrier_wait(B)             barrier_wait(B)
print(result)
                            result += …
```

FFT in SPLASH2

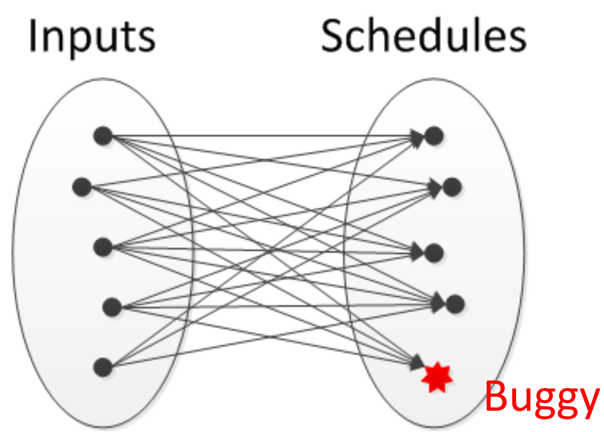
- *Schedule*: sequence of communication operations
- *Buggy schedule*: schedule that triggers concurrency bug

Challenges caused by nondeterminism



- Testing: less effective
- Debugging: more challenging

Challenges caused by too many schedules

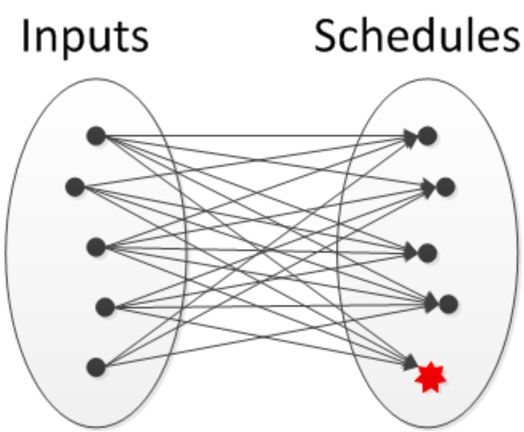


Traditional multithreading

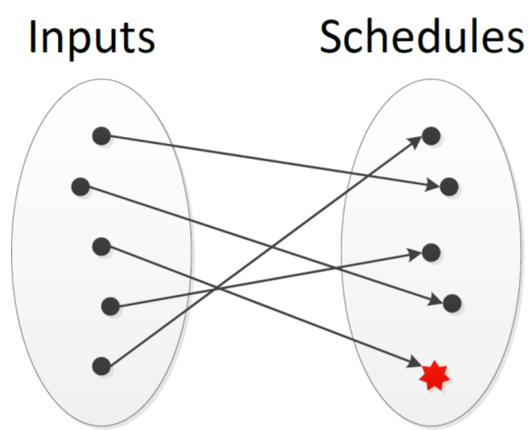
- m threads, k lock() ➔ more than (m!)^k schedules
- Even more schedules aggregated over all inputs
- Finding buggy schedules is like finding needles in a haystack

Determinism: neither sufficient nor necessary for reliability

Deterministic multithreading (DMT): one input → one schedule



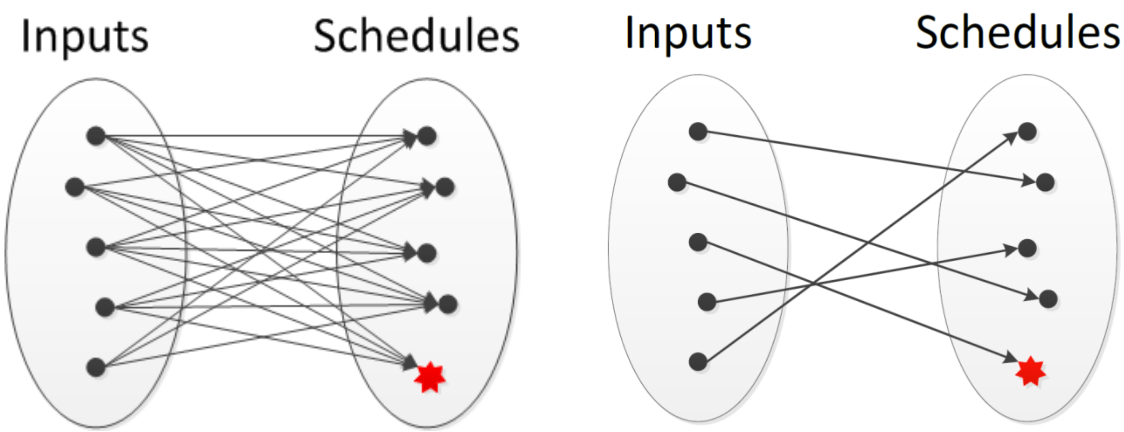
Traditional multithreading



Deterministic multithreading

- One testing execution validates all future executions on the same input
- Reproducing a concurrency bug requires only the input

Determinism: not sufficient

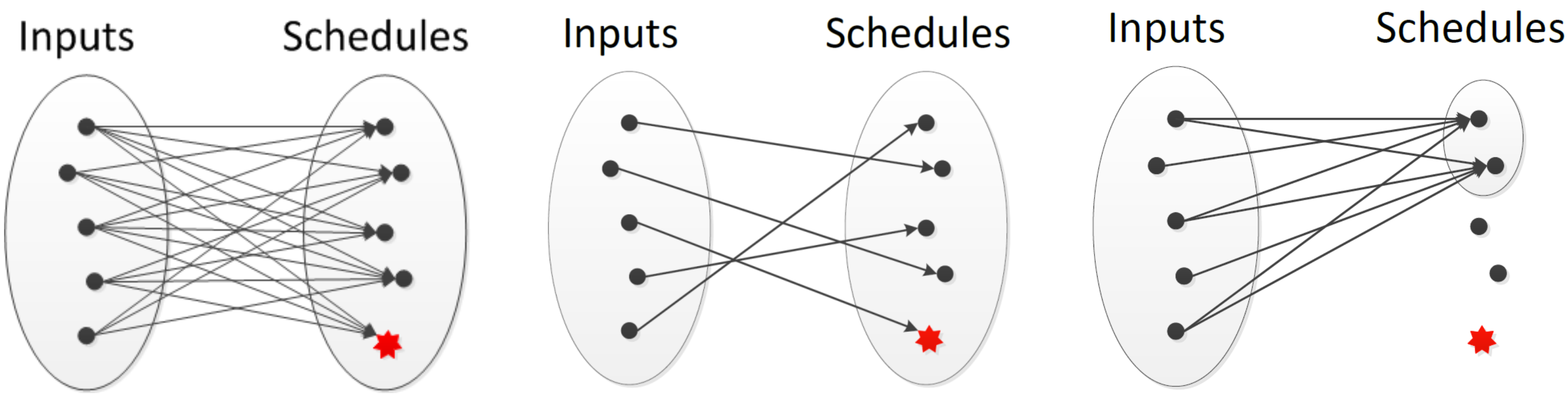


Traditional multithreading

Deterministic multithreading

- Determinism is a narrow property
  - Same input + same program ➔ same behavior
  - Input or program changes slightly? Unstable

Determinism: not necessary



Traditional multithreading

Deterministic multithreading

Stable multithreading

- Determinism is a binary property
  - Nondeterministic if one input ➔ n > 1 schedules
- Small n (e.g., 2) ➔ challenges caused by nondeterminism are easy to solve

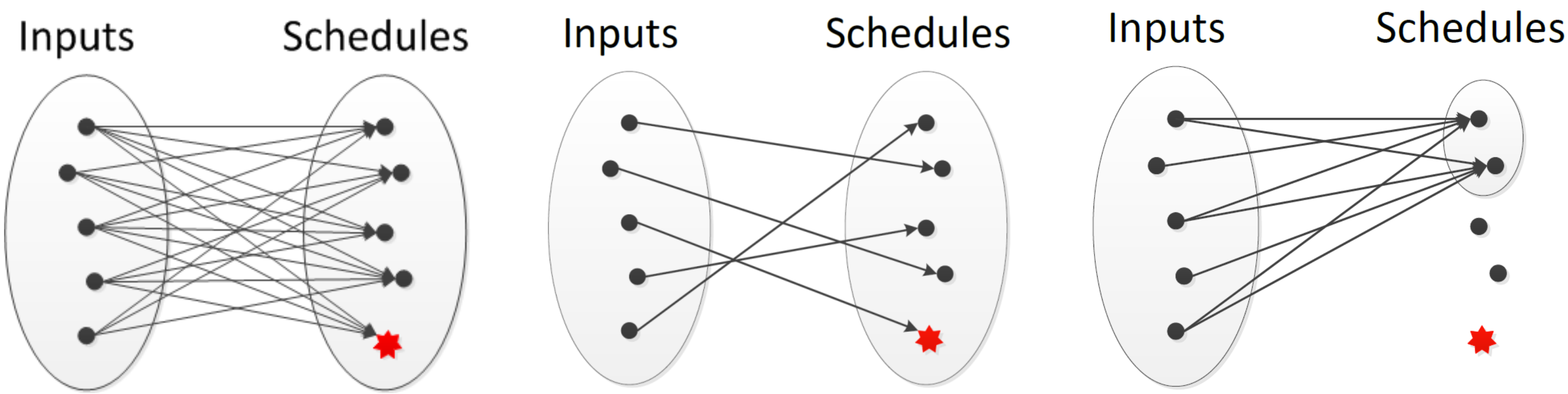
Improving reliability with stable multithreading

Are all exponentially many schedules necessary?

- Insight 1: for many programs, a wide range of inputs share the same equivalent class of schedules

- Insight 2: the overhead of enforcing a schedule on different inputs is low (e.g., 15%)

Stable multithreading (StableMT): all inputs → a small set of schedules



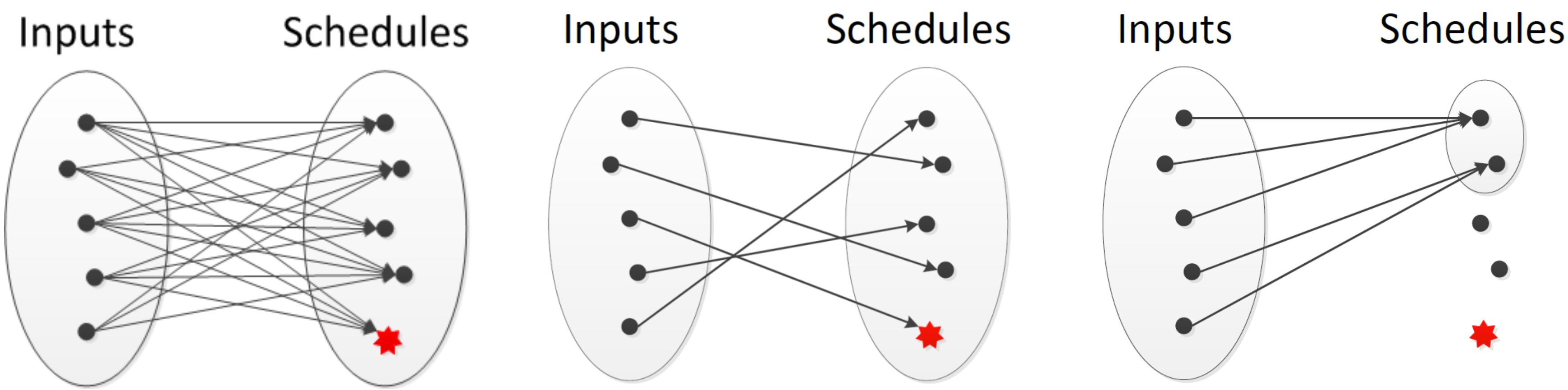
Traditional multithreading

Deterministic multithreading

Stable multithreading

- Vastly shrink the haystack → needles much easier to find
- Provide anticipated robustness and stability

StableMT and DMT: orthogonal



Traditional multithreading

Deterministic multithreading

Stable deterministic multithreading

Implementing and applying StableMT

- How can we compute the schedules to map inputs to?
  - Tern [OSDI 10]
- How can we enforce schedules deterministically and efficiently?
  - Peregrine [SOSP 11]
- How can we apply StableMT to effectively analyze multithreaded programs?
  - Schedule specialization [PLDI 12]

Conclusion

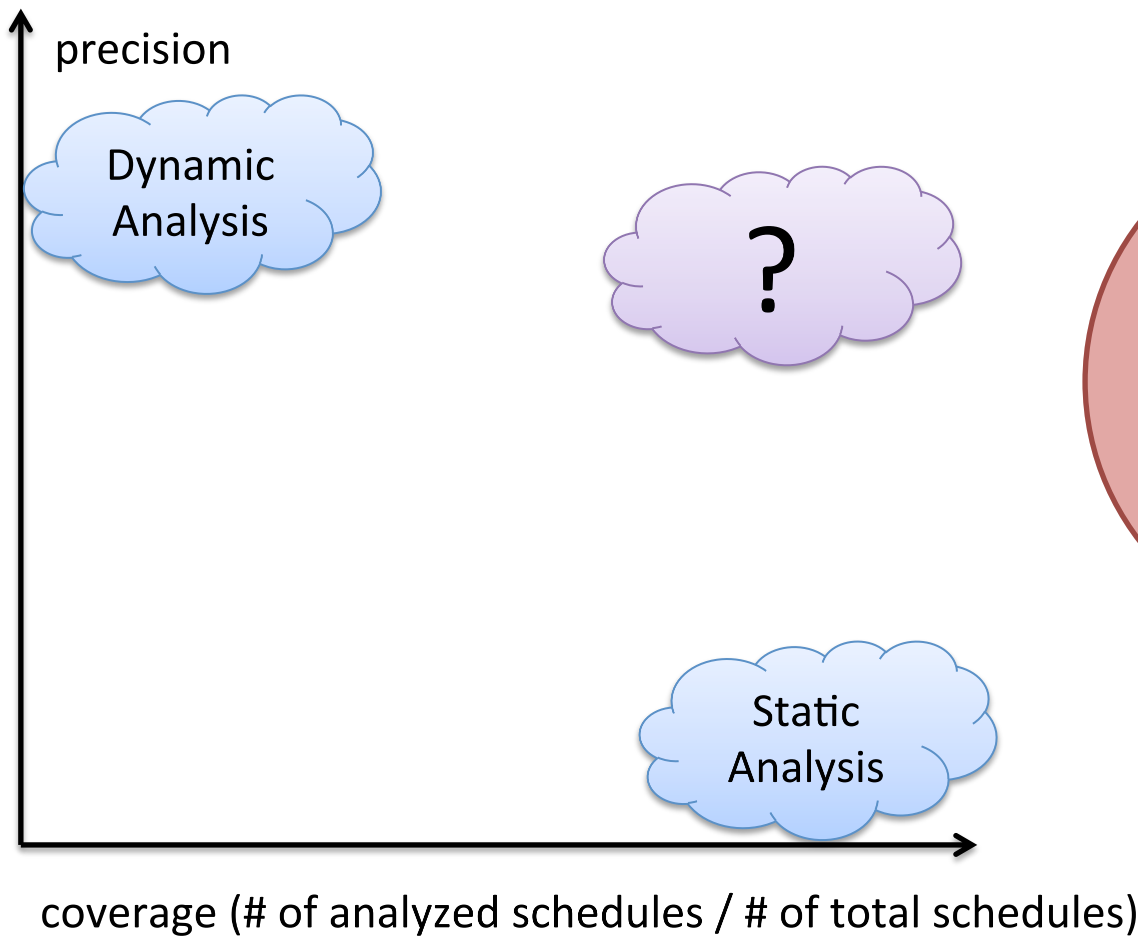
- Determinism: neither sufficient nor necessary for reliability
- StableMT: map all inputs to vastly reduced set of schedules, greatly improving reliability

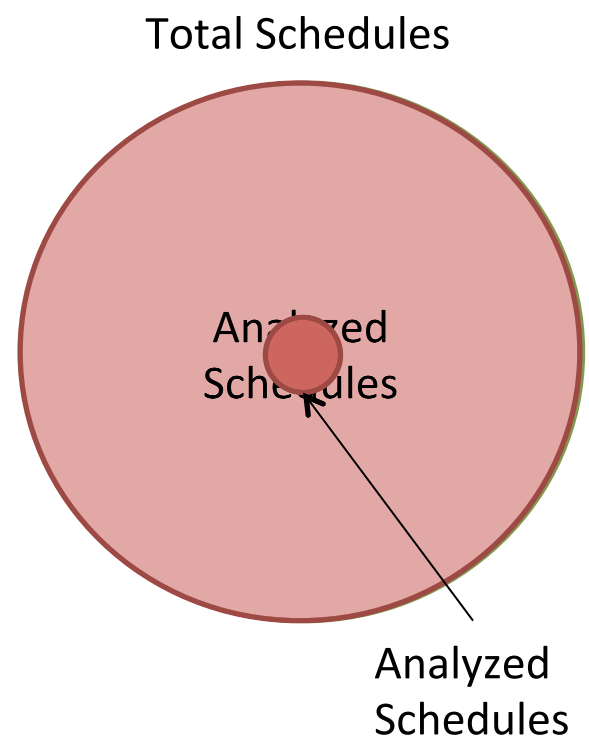
Future work

- Systems level: more efficient, lightweight, scalable
- Application level: more applications
- Conceptual level: StableMT programming languages, models, and methods

Applying StableMT to better analyze multithreaded programs

- Analyzing multithreaded programs: hard





Schedule Specialization [PLDI 12]

- Precision: Analyze the program over a small set of schedules.
- Coverage: Enforce these schedules at runtime.