

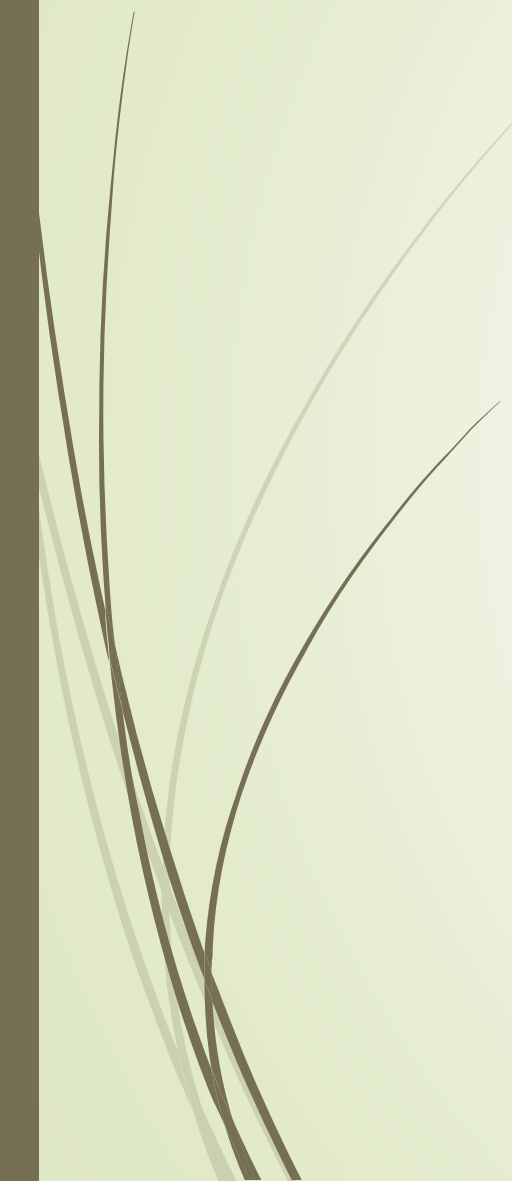


Automated Concurrency-Bug Fixing

Guoliang Jin, Wei Zhang, Dongdong Deng, Ben Liblit, Shan Lu
Concepts of Concurrent Computation seminar



Motivation

- ▶ Multi-core era
 - ▶ Bug detectors are already proposed
 - ▶ Fixing bugs is challenging
- 

```
graph TD; A[Bug Understanding] --> B[Fix-Strategy Design]; B --> C[Synchronization Enforcement]; C --> D[Patch Testing & Selection]; D --> E[Patch Merging];
```

Bug
Understanding

Fix-Strategy
Design

Synchronization
Enforcement

Patch Testing
& Selection

Patch
Merging

```
graph TD; A[Bug Understanding] --> B[Fix-Strategy Design]; B --> C[Synchronization Enforcement]; C --> D[Patch Testing & Selection]; D --> E[Patch Merging];
```

Bug
Understanding

Fix-Strategy
Design

Synchronization
Enforcement

Patch Testing
& Selection

Patch
Merging

```
graph TD; A[Bug Understanding] --> B[Fix-Strategy Design]; B --> C[Synchronization Enforcement]; C --> D[Patch Testing & Selection]; D --> E[Patch Merging];
```

Bug
Understanding

Fix-Strategy
Design

Synchronization
Enforcement

Patch Testing
& Selection

Patch
Merging

(a) Atomicity

(b) Order Violation

(c) Race

(d) Def-Use

Violation

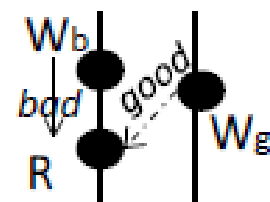
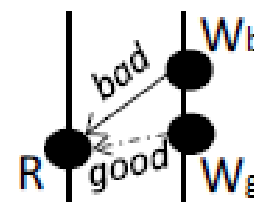
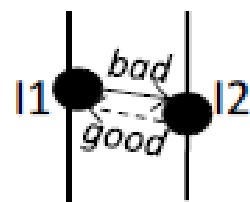
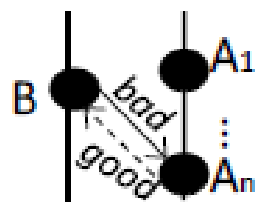
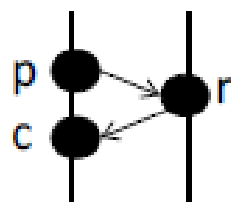
allA-B

firstA-B

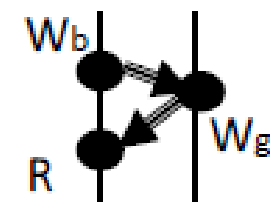
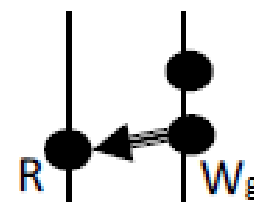
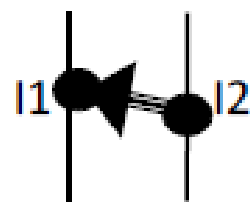
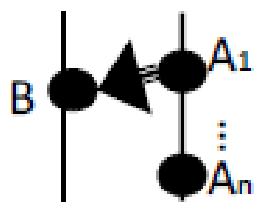
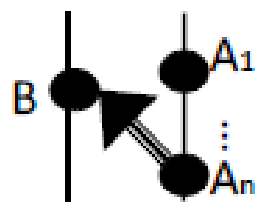
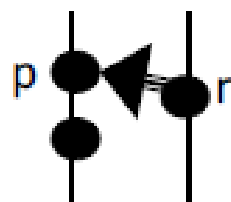
Remote-is-Bad

Local-is-Bad

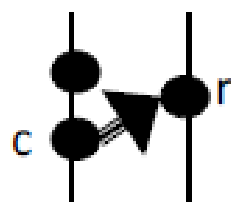
Reports:



Strategy (1):

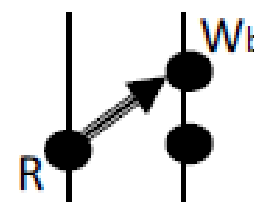
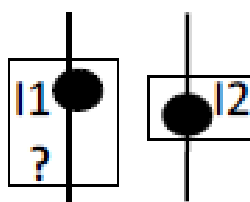


Strategy (2):



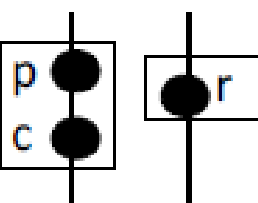
N/A

N/A



N/A

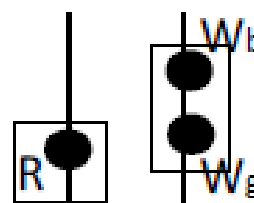
Strategy (3):



N/A

N/A

N/A



N/A



```
graph TD; A[Bug Understanding] --> B[Fix-Strategy Design]; B --> C[Synchronization Enforcement]; C --> D[Patch Testing & Selection]; D --> E[Patch Merging];
```

Bug
Understanding

Fix-Strategy
Design

Synchronization
Enforcement

Patch Testing
& Selection

Patch
Merging



Enforcing order relationship

- A instruction
- Signal thread
- S-create threads
- B instruction
- Wait thread
- W-create thread
- Call stack: $(f_0, i_0) \rightarrow (f_1, i_1) \rightarrow \dots \rightarrow (f_n, i_n)$



allA-B order

- ▶ locate places to insert signal operations in signal threads,
- ▶ locate places to insert signal operations in s-create threads,
- ▶ locate places to insert wait operations, and
- ▶ implement the signal and wait operations

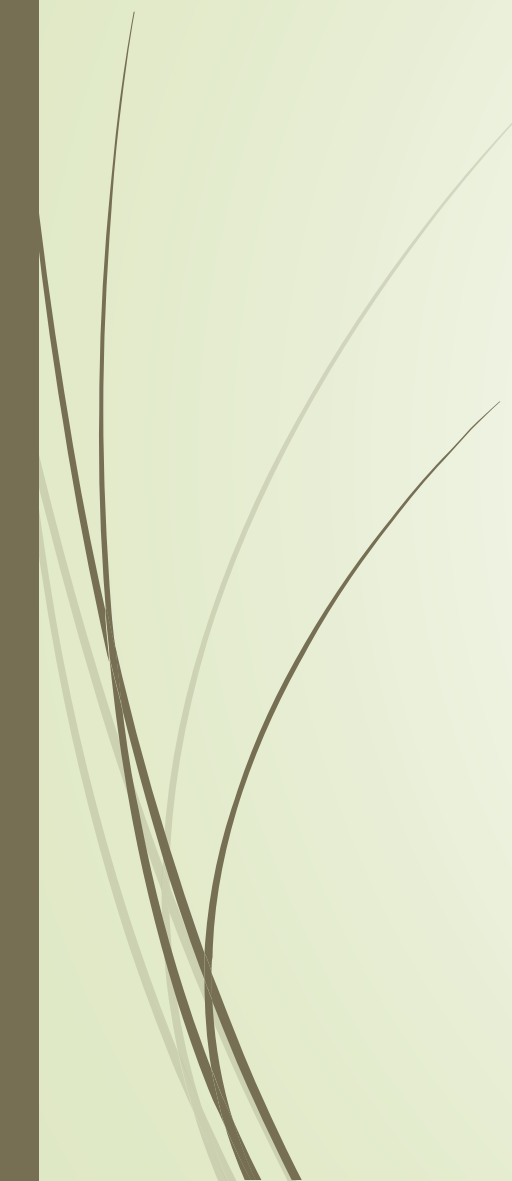


Finding Signal Locations in Signal Threads

- ▶ Analyse Control Flow Graph (CFG) of f_0
- ▶ Insert signal operation on CFG edge that goes from reaching to non-reaching node
- ▶ Continue down the call stack
- ▶ Can i_0 call f_1 multiple times?




allA-B order

- ▶ locate places to insert signal operations in signal threads,
 - ▶ locate places to insert signal operations in s-create threads,
 - ▶ locate places to insert wait operations, and
 - ▶ implement the signal and wait operations
- 



Implementing Wait and Signal Operations

- ▶ Track the number of threads that will perform signal operation
- ▶ Track how many threads have signaled already
- ▶ Allow a wait thread to proceed once all signals are done



firstA-B order

- ▶ Basic design
 - ▶ Signal right after A instruction
 - ▶ Wait right before B instruction
- ▶ Safety-net design
 - ▶ When program can no longer execute A, wait thread will continue
 - ▶ Inserts signal operations using *allA-B* algorithm

```
graph TD; A[Bug Understanding] --> B[Fix-Strategy Design]; B --> C[Synchronization Enforcement]; C --> D[Patch Testing & Selection]; D --> E[Patch Merging];
```

Bug
Understanding

Fix-Strategy
Design

Synchronization
Enforcement

Patch Testing
& Selection

Patch
Merging

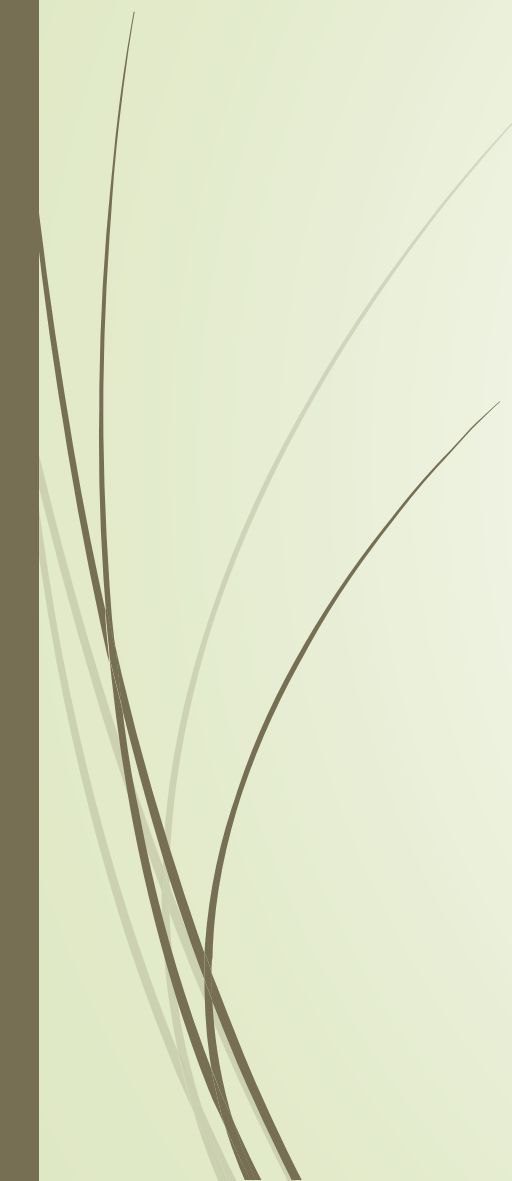


Correctness Testing

- Deadlock discovered by static analysis
- Failure in RTest
- Failure in GTest
- Timeout in RTest
- Failures of related patches



Patch selection

- ▶ Performance impact
 - ▶ Number of synchronization operations
 - ▶ Patch that can be merged
- 


```
graph TD; A[Bug Understanding] --> B[Fix-Strategy Design]; B --> C[Synchronization Enforcement]; C --> D[Patch Testing & Selection]; D --> E[Patch Merging];
```

Bug
Understanding

Fix-Strategy
Design

Synchronization
Enforcement

Patch Testing
& Selection

Patch
Merging



Patch merging guidelines

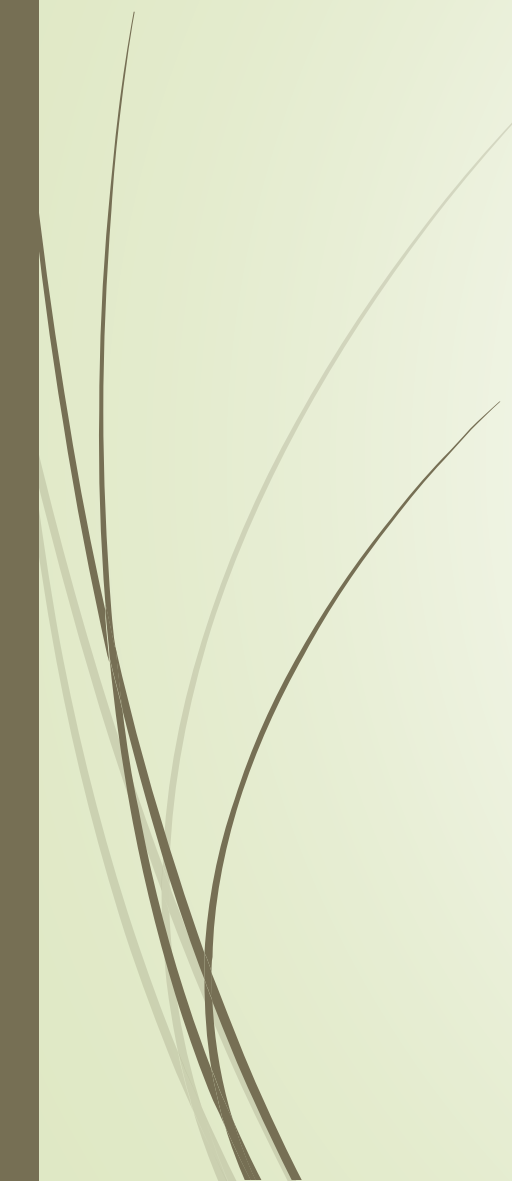
- The merged patch must have statically and dynamically fewer signal and wait operations than the unmerged patches.
- Each individual bug must still be fixed.
- Merging must not add new deadlocks.
- Merging should not cause significant performance loss.

Overall results

ID	Number of Bug Reports				Overall Patch Quality						Failure Rates		Overhead		# of CFix Sync Ops
	AV	OV	RA	DU	AV	OV	RA	DU	CFix	Manual	Original	CFix	CFix	Manual	
OB1	2	5 _a	4		✓	✓	✓		✓	✓	43%	0%	-0.3%	1.6%	5
OB2		1 _a				✓			✓	✓	65%	0%	-0.1%	3.6%	7
OB3	7	4 _f	10	4 _L	✓	✓	✓	✓	✓	✓	100%	0%	0.2%	0.0%	5
OB4	1	1 _f	2		✓	✓	✓		✓		97%	0%	0.5%		2
OB5	1		1		✓		✓		✓	✓	64%	0%	0.0%	0.0%	2
OB6	1	1 _f	2	1 _L	✓	✓	✓	✓	✓	✓	93%	0%	0.3%	-0.3%	2
OB7		1 _f				✓			✓	✓	97%	0%	0.2%		3
AB1	6		6		✓		✓		✓	✓	52%	0%	-0.9%	-0.4%	3
AB2	1		2	1 _R	✓		✓	✓	✓	✓	39%	0%	0.7%	0.5%	5
AB3	2		4	2 _R	✓		✓	-	✓	-	53%	0%	-0.0%	1.0%	9
AB4	1		2		✓		✓		✓	-	55%	0%	-0.5%	0.0%	3
AB5	4		5	1 _R	✓		✓	✓	✓	✓	68%	0%	-0.2%	0.4%	2
AB6	1		2	1 _R	✓		✓	✓	✓	✓	42%	0%	0.7%	0.5%	5

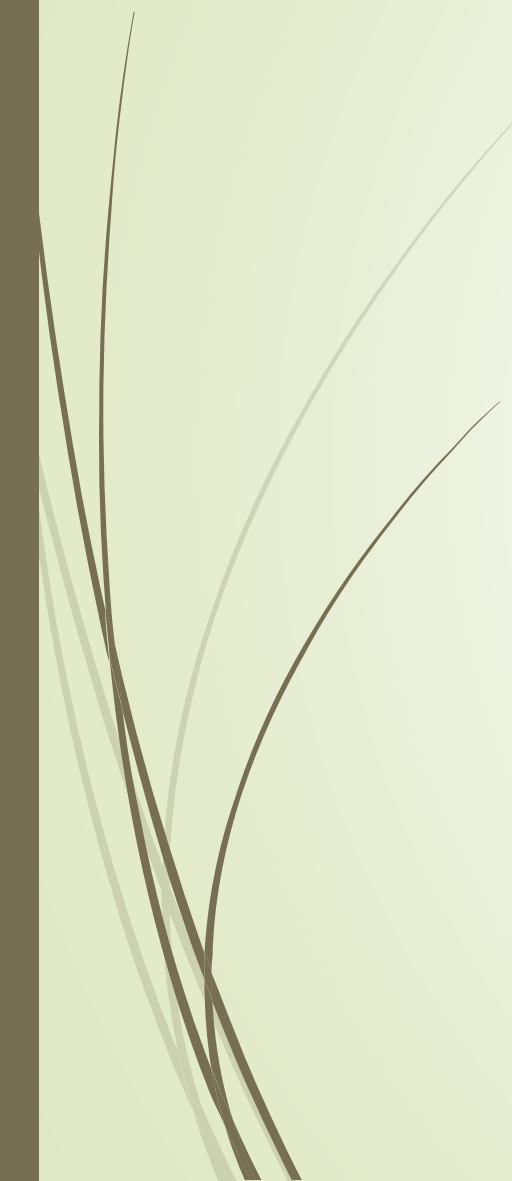


Limitations of CFix

- Only two different orderings
 - Bugs including shared loops
 - Some special scenarios...
- 



Conclusion

- Different bug detectors
 - Among first tools of this category
- 



Thank you!

Questions?

