Seminar Talk

Achieve High Synchronization Coverage in Testing Concurrent Programs

Hong, Ahn, Park, Kim, Harrold

KAIST, South Korea
and
Georgia Institute of Technology, USA
(published in July 2012)

20.3.2013

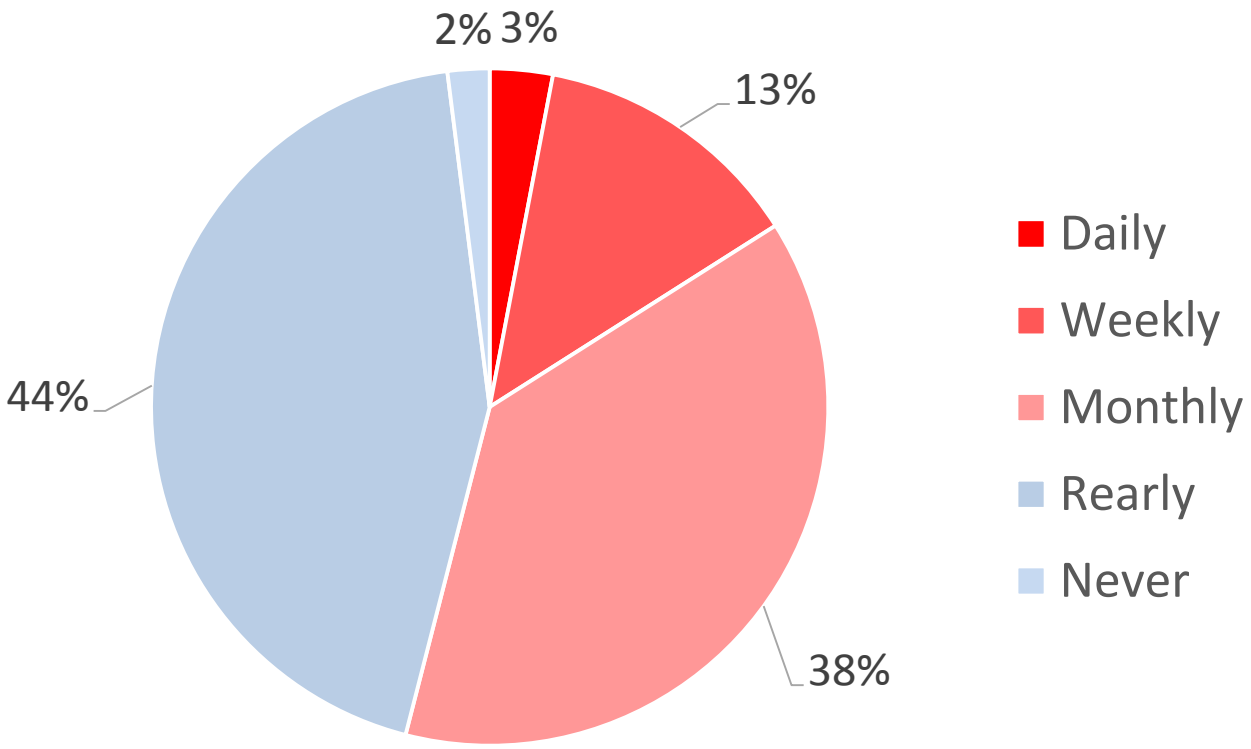
Benjamin Bucheli
buchelib@student.ethz.ch

Content

- Why test concurrent program in the first place?
- Metric: Synchronization coverage
- Achieving high synchronization coverage

How often do you detect/debug and fix concurrent bugs?

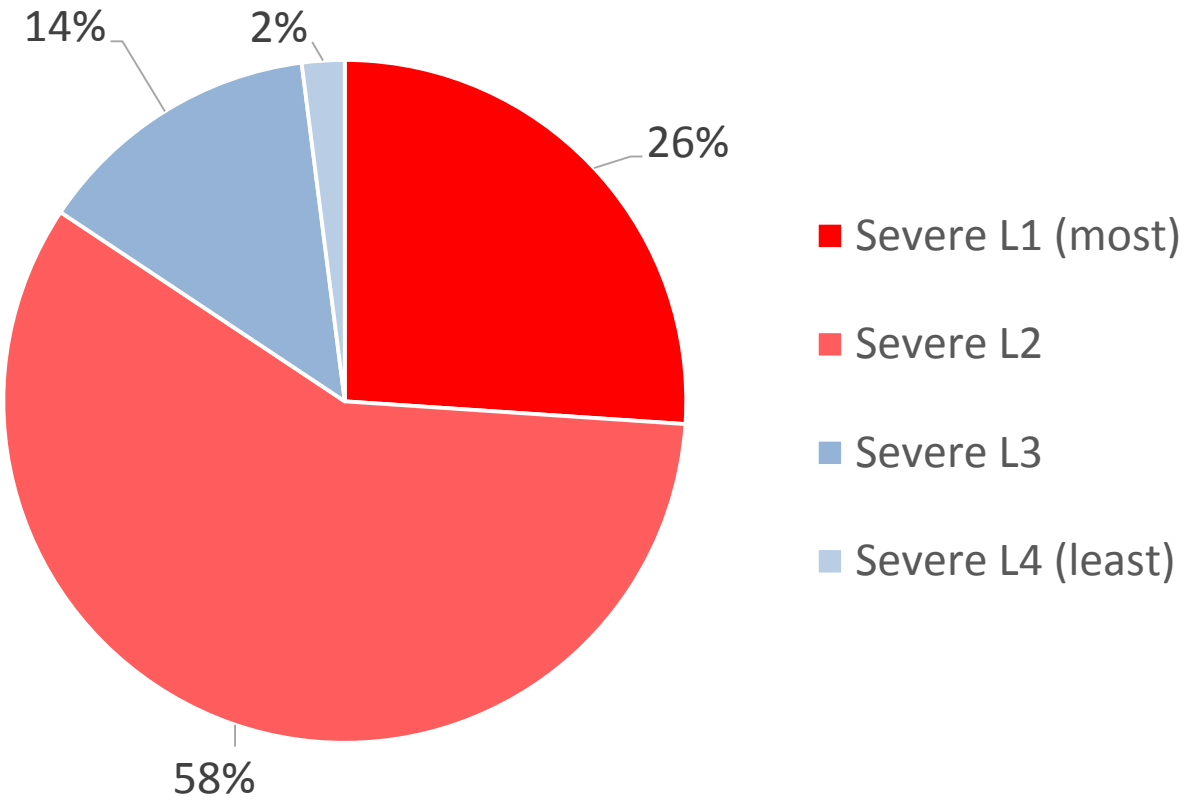
Source: [2]
An Exploratory Survey
Jan 2007 at Microsoft



➔ Over 50% of the respondants face concurrent bugs regularly.

How severe are the concurrent bugs?

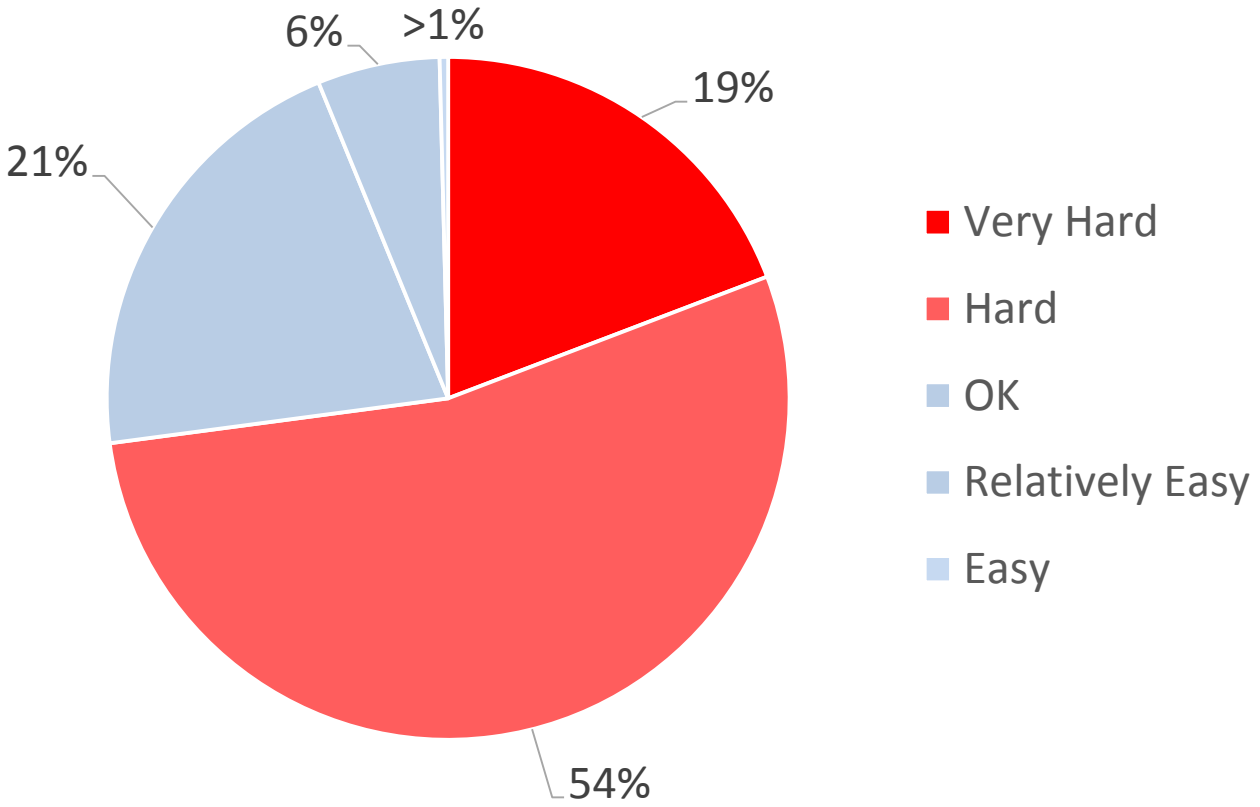
Source: [2]
An Exploratory Survey
Jan 2007 at Microsoft



➔ Majority of the concurrent bugs are rated severe.

How difficult is it to find concurrent bugs?

Source: [2]
An Exploratory Survey
Jan 2007 at Microsoft



➔ Concurrent bugs are often difficult to find.
It sometimes takes several days to locate them.

Coverage for Concurrent Programs

IBM introduced several coverage models for testing concurrent programs. One of which is called **synchronization coverage**.

```
10:  thread_1(){
11:   lock(m)
       …
20:   unlock(m)
21:  }
```

```
30:  thread_2(){
31:   lock(m)
       …
40:   unlock(m)
41:  }
```

```
50:  thread_3(){
51:   …
53:   lock(m)
       …
60:   unlock(m)
71:  }
```

Synchronization-pair:

<11, 31> <31, 11> <11, 53>
<53, 11> <31, 53> <53, 31>

Testing Concurrent Programs

	Test Type				
	Stress	Random	Pattern-Based	Systematic	TSA*)
Coverage (Sync.-pair)	-	★	★★	★★★	★★★
Limitation	– does not reveal any concurrent bugs	– does not reveal all bugs – Limited scalability	– focues on specific bugs	– Bad scalability	– Extra test-suite neccessary
Tools		ConTest	Eraser, Atomizer, CalFuzzer	CHESS, Fusion	

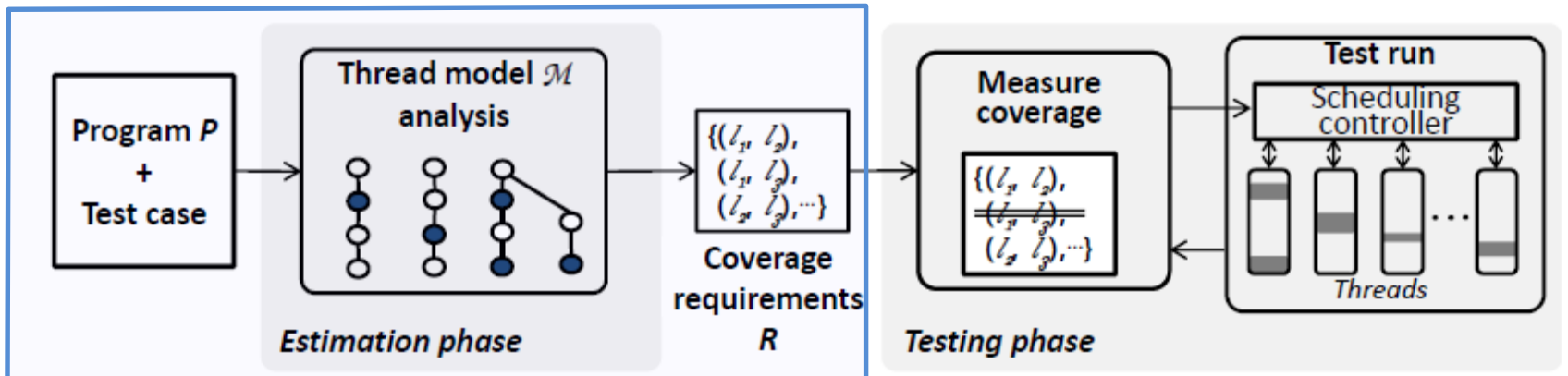
★★★ Close to max coverage

★★ Many Coverage

★ Some Coverage

*) TSA: Thread Scheduling Algorithm

Principle of TSA (Thread Scheduling Algorithm)



From the thread model, the SP are generated.

```
10:  thread_1(){
11:    lock(m)
       …
20:    unlock(m)
21:  }
```

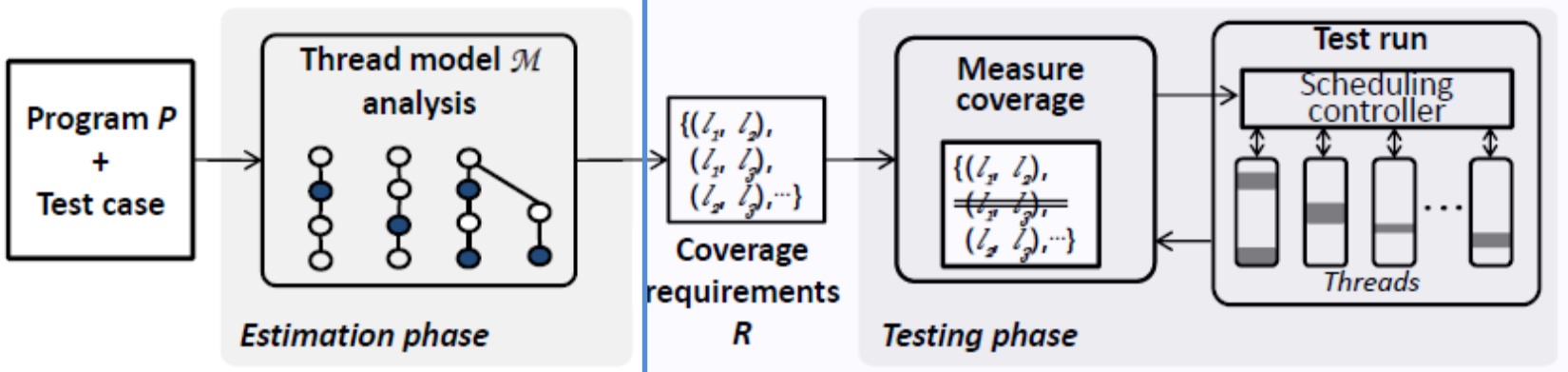
```
30:  thread_2(){
31:    lock(m)
         …
40:    unlock(m)
41:  }
```

```
50:  thread_3(){
51:    …
53:  lock(m)
     …
60:    unlock(m)
71:  }
```

<11, 31> <31, 11>
<11, 53> <53, 11>
<31, 53> <53, 31>
…

Coverage requirement R

Principle of TSA (Thread Scheduling Algorithm)



Invoke TSA ***before*** lock operation and ***after*** unlock operation. Either suspend or proceed thread based on 3 rules.

~~<11, 31>~~ ~~<31, 11>~~
~~<11, 53>~~ ~~<53, 11>~~
~~<31, 53>~~ ~~<53, 31>~~
...

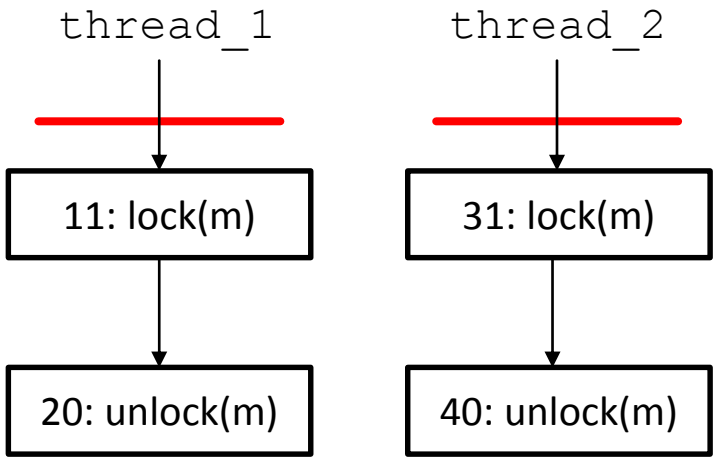
```
10:  thread_1(){
11:    lock(m)
       ...
20:    unlock(m)
21:  }
```

```
30:  thread_2(){
31:    lock(m)
       ...
40:    unlock(m)
41:  }
```

```
50:  thread_3(){
51:    ...
53:  lock(m)
       ...
60:    unlock(m)
71:  }
```

Details of the 3 Scheduling Rules

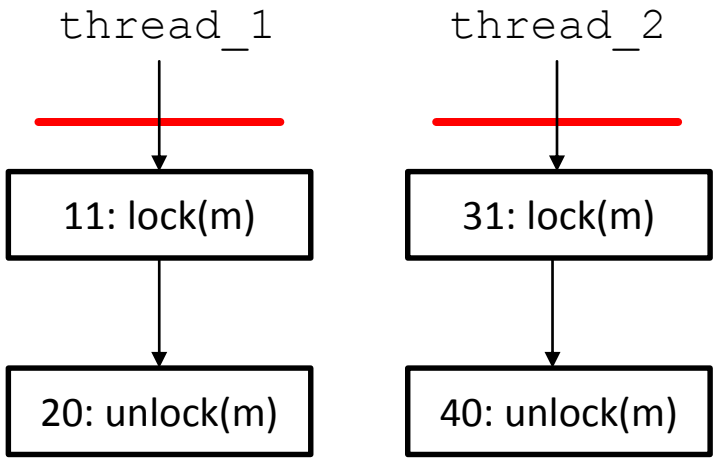
Rule 1: Proceed with thread that immediately satisfy uncovered SP.



<11, 31> <31, 11>
<11, 53> <53, 11>
<31, 53> <53, 31>
…

➔ Proceed with `thread_1` then `thread_2`

Rule 2: Proceed with thread that satisfy next uncovered SP with same lock.

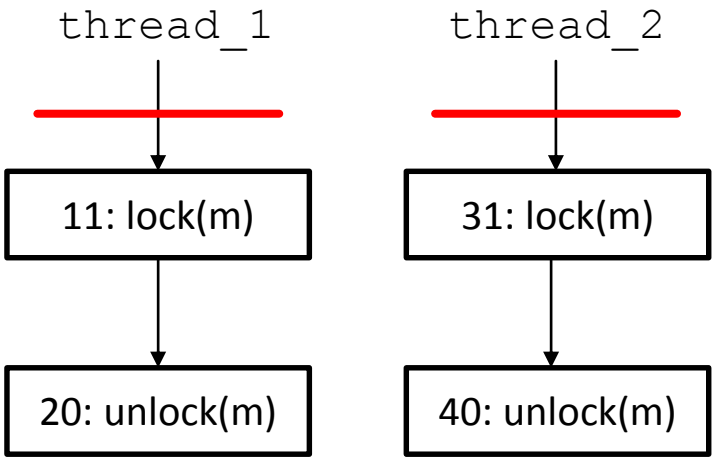


<11, 31> <31, 11>
<11, 53> <53, 11>
<31, 53> <53, 31>
…

➔ Proceed with `thread_2` then `thread_1`

Details of the 3 Scheduling Rules

Rule 3: Choose thread with smallest number of relevant coverage.



~~<11, 31>~~ ~~<31, 11>~~
~~<11, 53>~~ ~~<53, 11>~~
<31, 53> ~~<53, 31>~~
<11, 70> <31, 99>
<11, 66> <31, 25>
<11, 90> <99, 31>

There are 3 requirements for 11 left.

There are 4 requirements for 31 left.

➔ Proceed with `thread_1`

The approach of rule 3 is also called the *estimation based heuristic*.
This rule improves test performance dramatically.

Empirical Test Conditions

Subject program for empirical study:

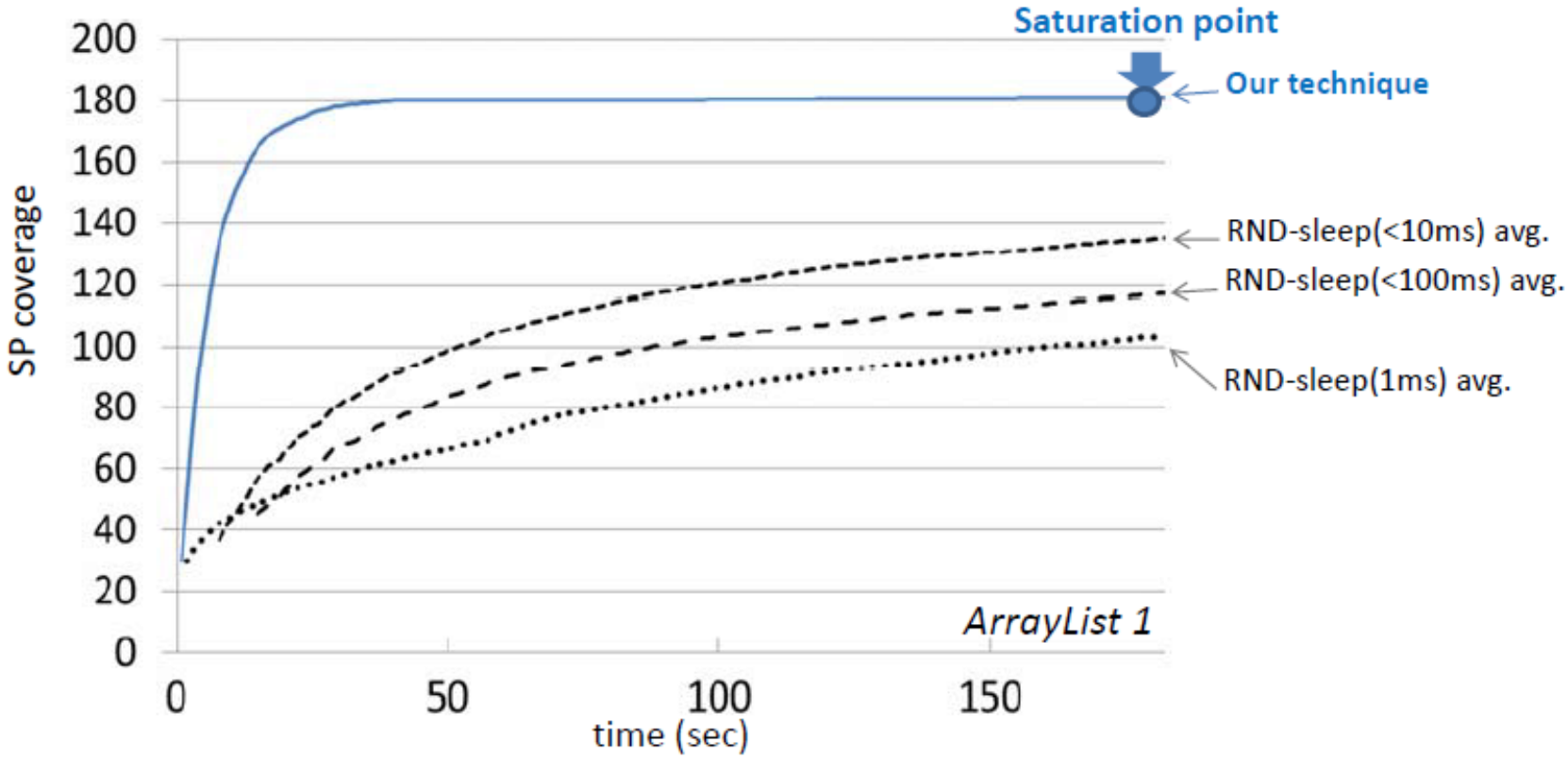
- Test programs out of Java libraries (see table).
- Execution of test 500 times for each technique
- Repetition of experiements 30 times (statistical reasons)

Type	Program	Lines of Code	# of Threads	# of Sync. stmts.
Java Library	ArrayList1	7712	26	69
	ArrayList2	7712	4	67
	HashSet1	9028	21	67
	HashSet2	9028	3	66
	HashTable1	11431	5	96
	HashTable2	11431	5	116
	LinkedList1	7375	26	67
	LinkedList2	7375	15	66
	TreeSet1	5669	21	69
	TreeSet2	5669	3	67
Java Server	cache4j	1922	10	128
	pool	5536	10	280
	VFS2	22981	6	116

Computer specification:

- Intel Core2 Duo 3.0 GHz;
- Sun Java SE 1.6.0
- Fedora Linux platform 9 (kernel 2.6.27)

Empirical Test Result (TSA vs. Random)



Source: [1]

➔ TSA resaches SP coverage *faster* and *higher.*

Side note: over 90% of decision are made by Rule 3

*SP: synchronization-pair

Critics about TSA

- Relationship between coverage and bug-detection?
- Technique generalizable?
- Extend other concurrent coverage criteria.

Summary

- Empirical studies show that TSA is more effective and efficient than random testing.
- TSA is scalable to large program (target code is not changed)
- Future works necessary to complete

END

Thank you