ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Effective Random Testing of Concurrent Programs

by Koushik Sen, UC Berkeley, CA, USA

Presented by: Marko Peric

Content

- Testing methods of concurrent programs
  - Traditional method and its problems
  - Model checking and the state-explosion problem
  - Partial order reduction technique
  - Randomized search
- Simple Randomized Algorithm
- RAPOS – Random Partial Order Sampling Algorithm
- Results

TESTING METHODS OF CONCURRENT PROGRAMS

Traditional testing

- How does it work?
  - Repeatedly execute the program in order to get different interleavings.

- Testing problems:
  - In a particular environment
  - Testing depends on the underlying operating system or VM.

- Testing is attractive because:
  - Inexpensive in comparison with model checking or verification.
  - Scales to very large programs.

Model checking

- How does it work?
  - Systematically controls the thread scheduler to get all behaviours of a program.

- Problems:
  - Can end up in a localized search of the state-space.
  - Does not scale with program size.
    - The number of possible interleavings often grows exponetially with the length of execution - STATE-EXPLOSION PROBLEM.

Partial order technique(1)

- Starts from the fact:
  - A number of interleavings are equivalent to each other because they correspond to different execution orders of various non-interacting (or independent) instructions from concurrent threads.
  - Different execution orders of non-inderacting instructions from concurrent threads result in the same overall final state.

- The result is:
  - If one execution order finds a defect => all equivalent execution orders will detect the defect.

Partial order technique(2)

- Happens-before relation defines a partial order over all the instructions executed during an execution.

- Concurrent executions with the same happens-before relation are equivalent and the set of them is a PARTIAL ORDER.

- Partial order advantages:
  - Addresses the state-explosion problem
  - Helping the model checkers to reduce the search space

Randomized search

- How does it work?
  - Picks up a random thread to execute at every execution point where a potential thread switch can happen.
- Why is it better than traditional testing?
  - Explicitly tries to control the scheduling of threads.
  - Explores wide variety of interleavings without getting stuck in a localized search.
- Problems:
  - Samples non-equivalent thread interleavings in a very non-uniform way => some partial orders are sampled more often than the others.

SIMPLE RANDOMIZED ALGORITHM

Simple Randomized Algorithm (1)

```
RandomExecutionSimple(){
        s := s0;
        while(Enabled(s) ≠ ∅) {
                take a random t out of Enabled(s);
                s := Execute(s,t);
        }
        if (Alive(s) ≠ ∅) {
        print “Deadlock Detected“;
        }
 }
```

Simple Randomized Algorithm (3)

Interleaving	Probability of sampling
1.	0.5
2.	0.25
3.	0.125
4.	0.0625
5.	0.03125

- One partial order with 0.96875 and the other with 0.03125.
- Ideally, sampling with probability of 0.5.

RANDOM PARTIAL ORDER SAMPLING ALGORITHM - RAPOS

RAPOS (1)

- Rapos(){

```
s := s0;
schedulable := Enabled(s);
while(Enabled(s) ≠ ∅){
        scheduled := RandIndependetSubset(schedulable);
        for each t ϵ scheduled
                s := Execute(s,t);
                schedulable := {t’ ϵ Enabled(s) | ∃t ϵ scheduled
                                such that t and t’ are dependent};
        if (schedulable = ∅)
                add a random element from Enabled(s) to schedulable;
}
If (Alive(s) ≠ ∅){
        print „Deadlock Detected“;
}
}
```

RAPOS (3)

Sets	Sets Prob. to be in scheduled	Interleav.	Interleav. Prob.
1. {y=1;}	0.25	5.	0.25
2. {if(x==4) assert(false);}	0.25	1.	0.25
3. {y=1, if(x==4) assert(false);}	0.5	1. or 2.	0.5

Sets assigned to scheduled after the 1st iteration

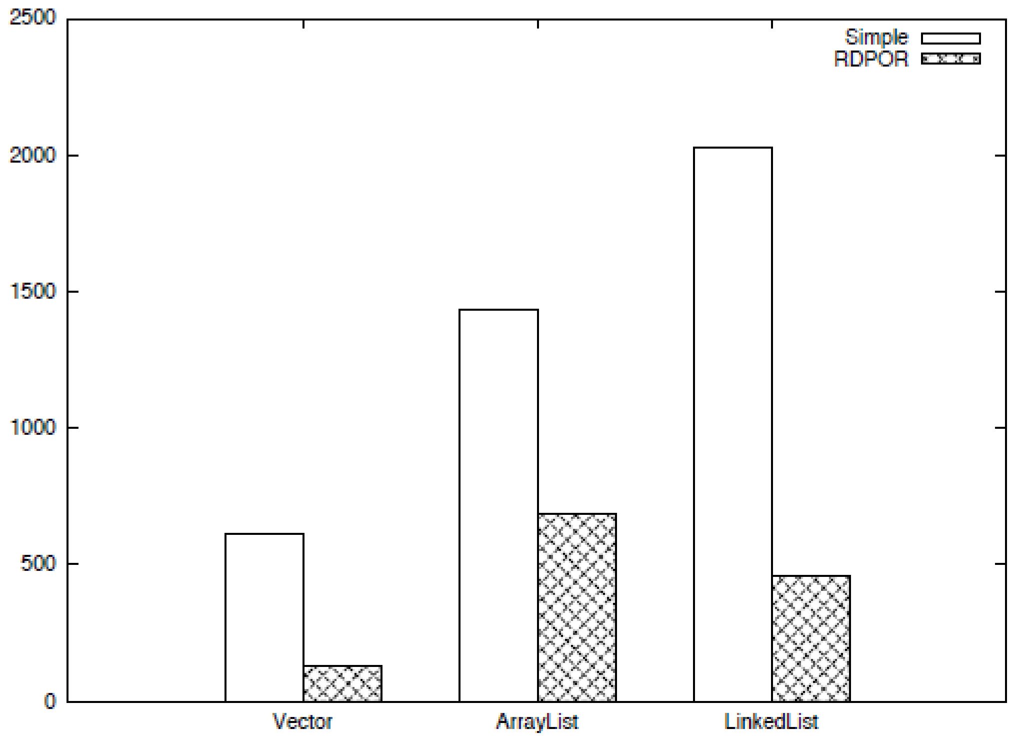
- The two partial orders will be sampled with probability 0.25 and 0.75.
- Assert violation with 0.25. Higher than SRA, 0.03125.

Results (1)

Program	SLOC	# of Threads	Total # of Executions	# of distinct Partial Orders		Column 6/Column 5
				Simple Algorithm	RAPOS	
ArrayList	-	7	300	71	247	3.47
boundedbuffer	141	4	300	118	171	1.45
HashSet	-	7	300	42	230	5.48
LinkedHashSet	-	7	300	48	230	4.79
LinkedList	-	7	300	58	259	4.47
moldyn	1291	11	100	19	48	2.53
montecarlo	3557	11	100	4	75	18.75
philo	91	3	2000	1124	1552	1.38
pipeline	119	8	300	38	156	4.11
raytracer	1859	11	100	22	44	2.00
TreeSet	-	7	300	78	240	3.08
Vector	-	7	300	105	256	2.44

Number of Partial Orders Sampled after a Fixed Number of Executions

Results (2)

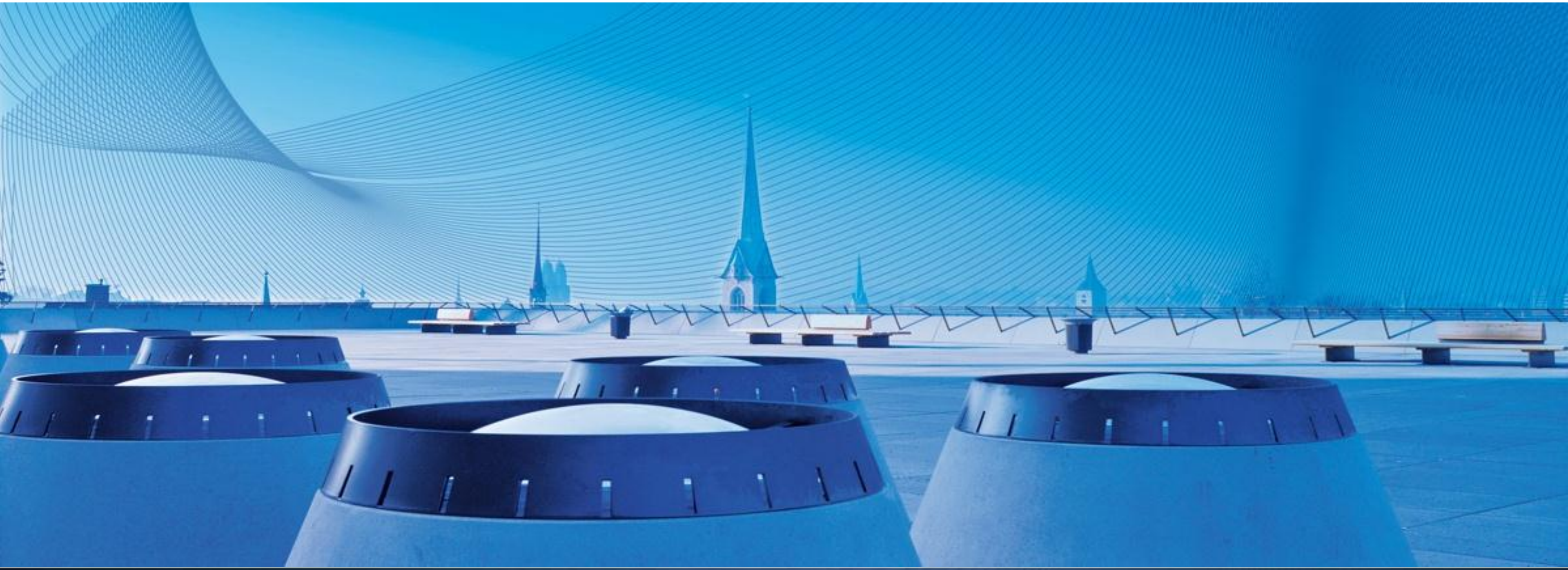


Expected Number of Executions Required to Detect a Defect

RAPOS vs SRA

- Higher number of partial orders with fixed number of executions
- Detecting defects after significantly less number of executions
- More uniform probability of sampling a partial order

Questions?



Simple Randomized Algorithm (2)

- Initially x = 0 and y = 0

Thread 1:	**Thread 2:**
y = 1;	if(x==4) assert(false);
y = 2;	
y = 3;	
x = 4;	

If (x==4) assert(false); y=1; y=2; y=3; x=4;

y=1; If (x==4) assert(false); y=2; y=3; y=4;

y=1; y=2; If (x==4) assert(false); y=3; y=4;

y=1; y=2; y=3; If (x==4) assert(false); y=4;

y=1; y=2; y=3; y=4; If (x==4) assert(false);