Efficient and Scalable Incremental Verification of Multithreaded Programs

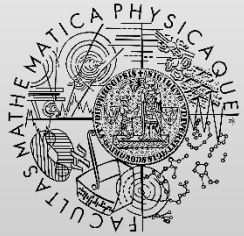
By Checking Just Pairs of Threads

http://d3s.mff.cuni.cz

Department of Distributed and Dependable Systems

D3S

Pavel Parízek

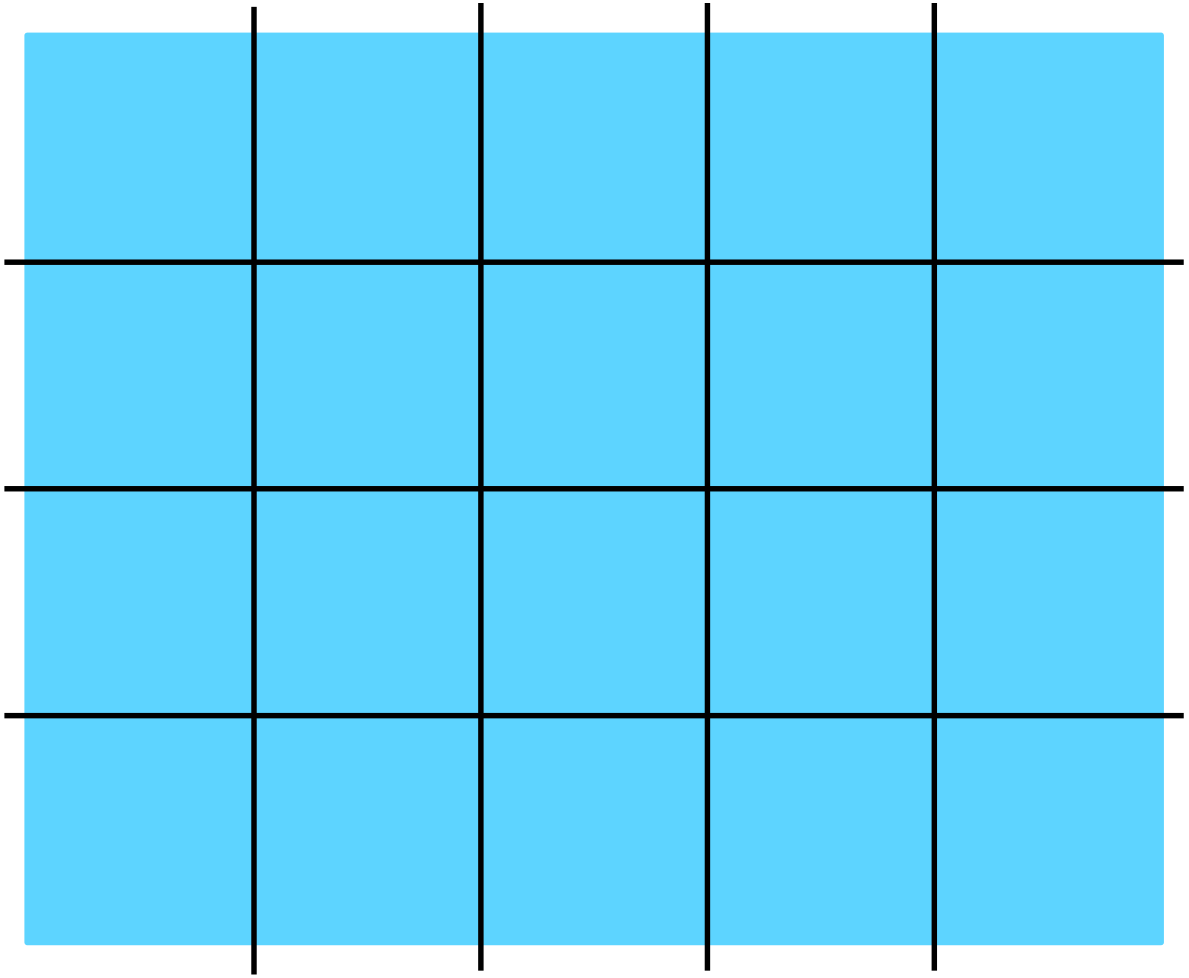


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

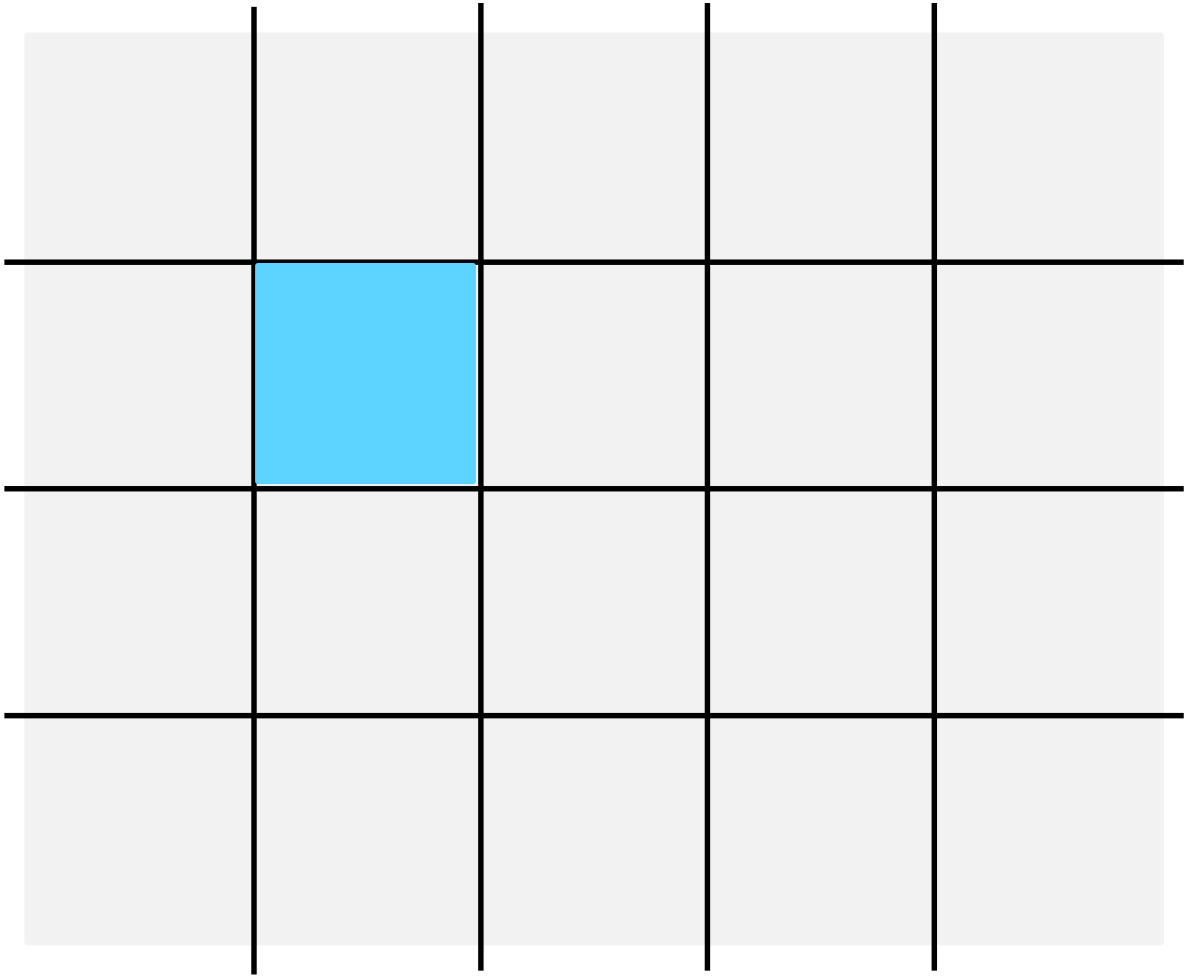
Goal: efficient and scalable verification

Goal: efficient and scalable verification

Incremental verification ?



Checking just recent code modifications



Our focus: multithreaded programs

```
T1 : o.f = x ; t2a = start T2 ; t2b = start T2

T2 : evaluate(o)

T3 : y = p.g ; z = r.h ; w = y + z
```

Department of Distributed and Dependable Systems D3S

Our focus: multithreaded programs

Challenge: too many possible thread interleavings

Goal: explore just small number of affected thread interleaving to report concurrency errors fast

Department of Distributed and Dependable Systems
D3S

Pairwise approach

- Exploring possible thread interleavings just for
  - Specific pairs of threads
  - Modified code fragment

Pairwise approach

- Exploring possible thread interleavings just for
  - Specific pairs of threads
  - Modified code fragment

- Considering all pairs (T_{mod}, T_j)
  - Thread with modified code
  - Any other program thread

Pairwise approach

- Ex
  -
  -
- Co
  -
  -

Checking interactions of recently modified code with every other thread

Pairwise approach – example

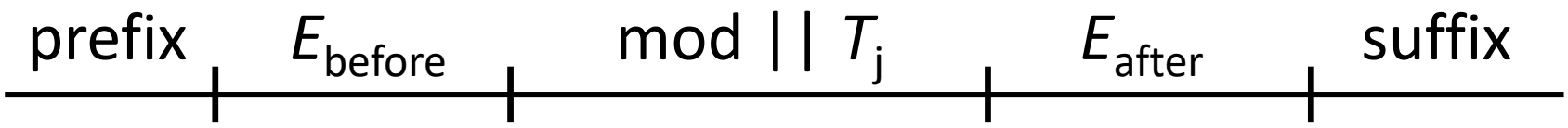
```
T1 : o.f = x ; t2a = start T2 ; t2b = start T2

T2 : evaluate(o) ; print(o.f)

T3 : y = p.g ; z = r.h ; w = y + z
```

Pairwise approach – example

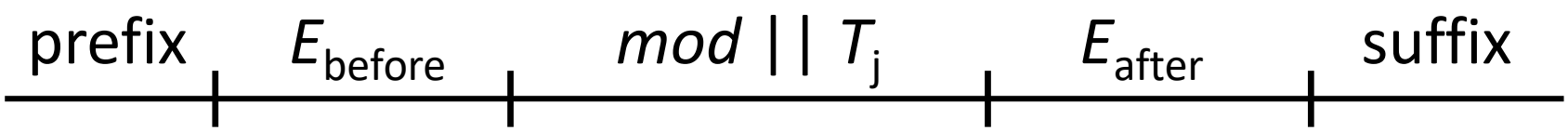
```
T1 : o.f = x ; t2a = start T2 ; t2b = start T2
T2 : evaluate(o) ; print(o.f)
T3 : y = p.g ; z = r.h ; w = y + z
```

Pairs of threads: (T2 T1), (T2 T2), (T2 T3)

Algorithm: explored traces



Algorithm: explored traces



- prefix: actions happening strictly before *mod*
- suffix: actions happening strictly after *mod*

- $E \equiv$ set of events **not** strictly before/after *mod*

Properties of the algorithm

1) Ability to find all concurrency errors that involve *N* threads also for *N* > 2

Properties of the algorithm

1) Ability to find all concurrency errors that involve *N* threads also for *N* > 2
   - Proof: using the concept of right and left movers for independent (thread local) actions within the middle part (mod || T_j)

Properties of the algorithm

1) Ability to find all concurrency errors that involve *N* threads also for *N* > 2

2) Covering all thread interleavings within the full state space of the most recent program version

- Proof: inductive approach over incremental source code modifications during the whole process of software development

Implementation

- Backend verification tool
  - Java Pathfinder
    - https://github.com/javapathfinder/jpf-core
- Continuously under development
  - https://github.com/d3sformal/incverif-pairwise

Experimental evaluation

- Benchmarks: multithreaded Java programs
- Approach: simulating incremental development of the subject programs
  - Scenarios: inserting/deleting various code fragments
- Configurations
  - Fast search for concurrency errors (“bug finding”)
  - Verifying safety (exploring all thread interleavings)
  - Full traversal of the whole state space

Department of Distributed and Dependable Systems D3S

Results: fast bug-finding

program	pairwise incremental verification		full verification	
	time: avg ± dev	timed out	time: avg ± dev	failed
Alarm Clock	0.25 ± 0.22 s	0 %	1.20 ± 2.60 s	0 %
Prod-Cons	0.14 ± 0.08 s	0 %	0.16 ± 0.04 s	0 %
RAX Extended	0.17 ± 0.23 s	0 %	0.15 ± 0.05 s	1.1 %
Rep Workers	4.67 ± 12.14 s	3.4 %	5.31 ± 29.30 s	2.8 %
SQR	0.09 ± 0.17 s	0 %	0.83 ± 0.92 s	0 %
Cache4j	8.14 ± 15.06 s	0.84 %	32.46 ± 283.76 s	0 %
Elevator	7.92 ± 16.14 s	1.67 %	12.19 ± 71.40 s	1.0 %
QSort MT	1.01 ± 2.27 s	1.25 %	0.61 ± 0.41 s	0.7 %
jPapaBench	15.96 ± 25.93 s	0 %	28.52 ± 178.98 s	0 %

Results for bug-finding – discussion

program	pairwise incremental verification		full verification	
	time: avg ± dev	timed out	time: avg ± dev	failed
Alarm Clock	0.25 ± 0.22 s	0 %	1.20 ± 2.60 s	0 %
Prod-Cons	**0.14 ± 0.08 s**	**0 %**	**0.16 ± 0.04 s**	**0 %**
RAX Extended	0.17 ± 0.23 s	0 %	0.15 ± 0.05 s	1.1 %
Rep Workers	4.67 ± 12.14 s	3.4 %	5.31 ± 29.30 s	2.8 %
SQR	0.09 ± 0.17 s	0 %	0.83 ± 0.92 s	0 %
Cache4j	**8.14 ± 15.06 s**	**0.84 %**	**32.46 ± 283.76 s**	**0 %**
Elevator	7.92 ± 16.14 s	1.67 %	12.19 ± 71.40 s	1.0 %
QSort MT	1.01 ± 2.27 s	1.25 %	0.61 ± 0.41 s	0.7 %
jPapaBench	**15.96 ± 25.93 s**	**0 %**	**28.52 ± 178.98 s**	**0 %**

Department of Distributed and Dependable Systems D3S

Results: checking safety

program	pairwise incremental verification		full verification	
	time: avg ± dev	timed out	time: avg ± dev	failed
Alarm Clock	0.25 ± 0.24 s	0 %	249.54 ± 143.48 s	0 %
Prod-Cons	0.16 ± 0.11 s	0 %	6.33 ± 1.82 s	0 %
RAX Extended	0.17 ± 0.24 s	0 %	12.71 ± 35.15 s	1.1 %
Rep Workers	41.32 ± 128.76 s	4.6 %	4494.43 ± 2859.57 s	14.7 %
SQR	0.09 ± 0.16 s	0 %	352.02 ± 74.26 s	0.9 %
Cache4j	46.18 ± 128.60 s	2.1 %	4936.50 ± 1987.80 s	2.7 %
Elevator	**27.38 ± 54.99 s**	**0.6 %**	**102.96 ± 394.04 s**	**88.2 %**
QSort MT	23.72 ± 92.62 s	3.8 %	735.84 ± 360.06 s	5.6 %
jPapaBench	**18.41 ± 32.01 s**	**0 %**	**8.19 ± 168.26 s**	**89.3 %**

Department of Distributed and Dependable Systems D3S

Future work

- Additional experiments with large realistic programs (e.g., from the DaCapo suite)
- Automated identification of thread pairs that do not have to be verified (certainly safe)