UNIVERSITÄT SALZBURG

A Randomized Scheduler with Probabilistic Guarantees of Finding Bugs

by

Sebastian Burckhardt, Pravesh Kothari, Madanlal Musuvathi, Santosh Nagarakatte

Multicore Acceleration of Prioriy-Based Schedulers for Concurrency Bug Detection

by

Santosh Nagarakatte, Sebastian Burckhardt, Milo M. K. Martin, Madanlal Musuvathi

Präsentation

Manuel Maier

Seminar Concurrency and Memory Management

Department of Computer Sciences

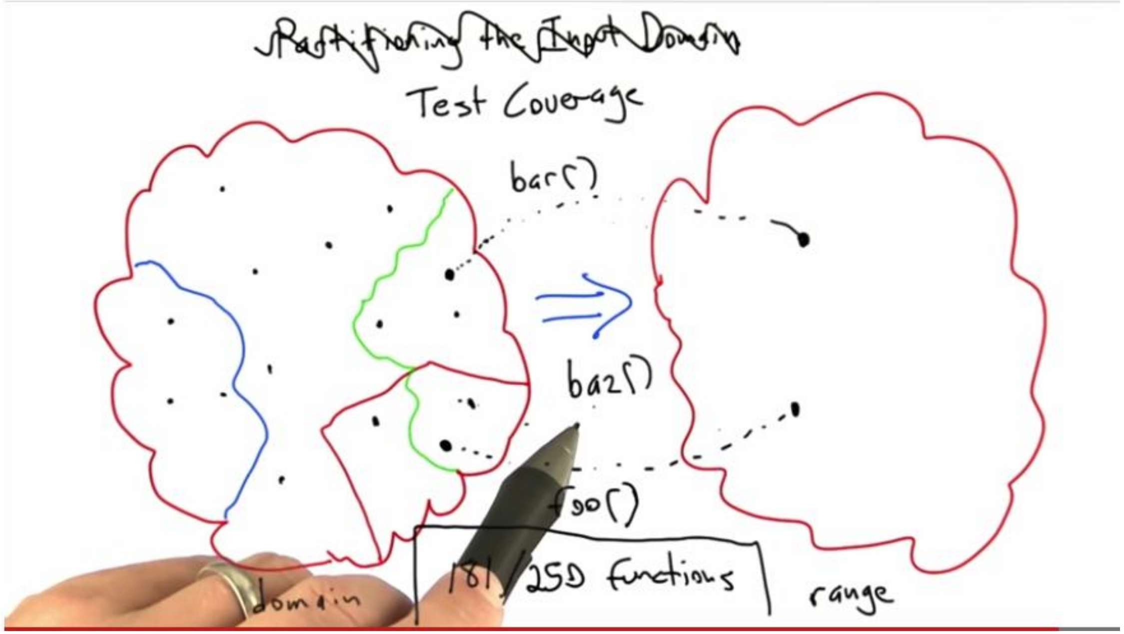
Seminarleiter: Univ.-Prof. Dr. Christoph Kirsch

Introduction

- What is the problem?
  - Testing concurrency phenomena
  - How to find buggy code paths/code schedules?
- State of the art
  - Stress testing
  - Heuristic-directed testing
  - Systematic scheduling
  - Randomized scheduling(e.g.: PCT)
- What is a radomized scheduler?
  - PCT provides guaranteed probability of $1/(nk^{d-1})$
  - With n number of threads, k number of instructions and d depth.

Exkurs (Test) Coverage

- What is „Test Coverage“?



- What is the problem with coverage concerning Concurrency Testing?
- Image taken from [3]

Controlled scheduling

- Schedule selection
  - Choose the one schedule that causes an error
- Controlled scheduling
  - Less common interleavings.
  - Os threads controlled in User Mode
  - A scheduling policy → Determine the thread that makes progress

Needlepoint

- To test efficacy of previously proposed concurrency bug detectors.
- Mechanisms
  - Instrumenting synchronization operations
  - Blocking information
  - Fairness and starvation freedom
- Policies

UNIVERSITÄT SALZBURG

Previously proposed concurrency bug detectors

- Random sleep(RS)
- Preemption always(PA)
- Preemption bounding(PB)
- Atom fuzzer(AF)
- PCT

Previously proposed concurrency bug detectors(2)

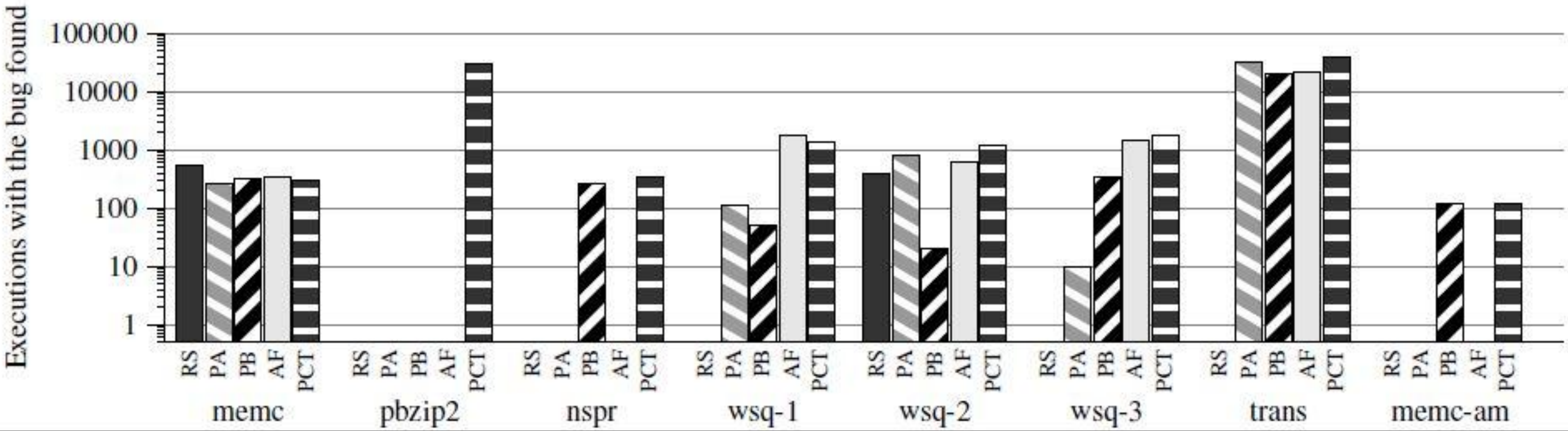


Figure 1. Bug detection abilities on common concurrency bugs for five different scheduling policies described in Section 3.2: Random Sleep (RS), Preemption Always (PA), Preemption Bounding (PB), AtomFuzzer (AF), and Probabilistic Concurrency Testing (PCT).

- Image taken from [2]

UNIVERSITÄT SALZBURG

Typical concurrency bugs

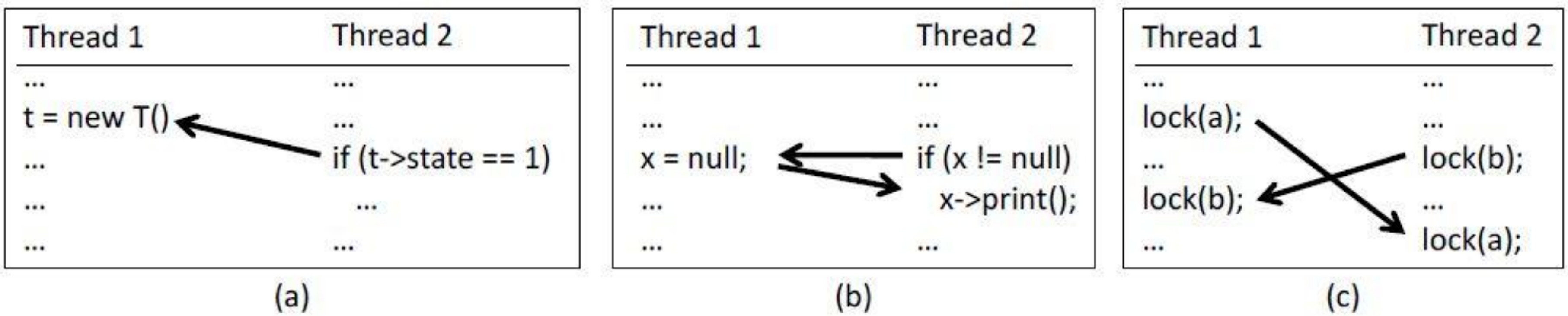


Figure 1. Three typical concurrency bugs, and ordering edges sufficient to find each. (A) This ordering bug manifests whenever the test by thread 2 is executed before the initialization by thread 1. (B) This atomicity violation manifests whenever the test by thread 2 executed before the assignment by thread 1, and the latter is executed before the method call by thread 2. (C) This deadlock manifests whenever thread 1 locks a before thread 2, and thread 2 locks b before thread 1.

- Image taken from [1]

UNIVERSITÄT SALZBURG

Bug Depth(d)

- Metric to classify concurrency bugs
- Minimum number of ordering constraints sufficient to find the bug.
  - Ordering bugs → Depth 1
  - Atomicity violations → Depth 2
  - Deadlocks(circular acquisition) → Depth n(n Threads)
- Preemption bound(d – 1):
  Smallest number of preemptions to find bug

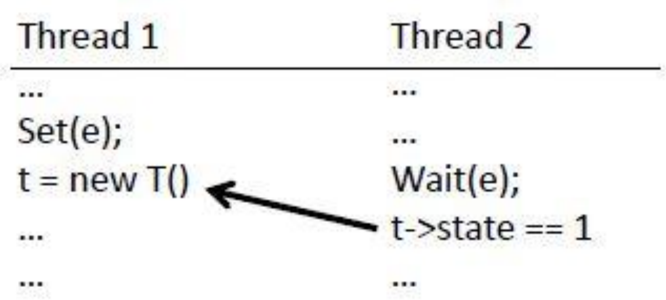


Figure 3. A variation of the example in Fig. 1(a), with the same bug depth of $d = 1$. Unlike in the other example, however, this bug requires Thread 1 to be preempted right after the instruction that sets the event e, and thus has a preemption bound of 1.

- Image taken from [1]

UNIVERSITÄT SALZBURG

Bug Depth(2)

Program	Lines of code	Schedule points	Threads	Bug type	Bug depth
Pbzip2	15,188	1210	3	Ordering violation	2
Memc	11,182	845	4	Atomicity violation	2
t+15		2300			
t+25		3400			
t+35		3900			
am		79132			
WSQ-1	541	1916	4	Atomicity violation	3
WSQ-2		1086			2
WSQ-3		1717			2
Trans	33,622	38118	2	Ordering violation	1
NSPR	1,100	5361	3	Deadlock	2

Table 1. Concurrency bugs used for NeedlePoint's evaluation that we obtained from prior research [5, 10, 16, 27, 28]. WSQ is an implementation of the work stealing queue with lock free data structures. WSQ-1, WSQ-2, and WSQ-3 are the three distinct concurrency bugs in the WSQ implementation. Memc is the memcached daemon. Trans is the Transmission BitTorrent client.

- Image taken from [2]

UNIVERSITÄT SALZBURG

Naive Randomization

- Flip coin each step

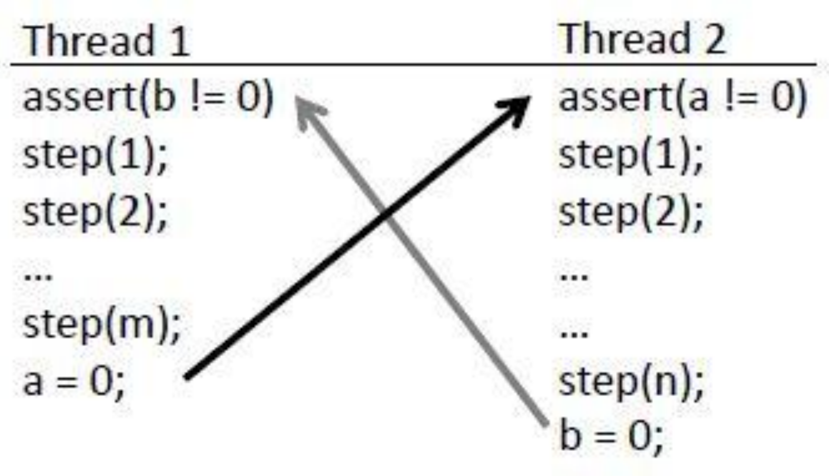


Figure 5. A program with two bugs of depth 1 that are hard to find with naive randomized schedulers that flip a coin in each step. PCT finds both these bugs with a probability 1/2.

- Image taken from [1]

PCT – Probabilistic Concurrency Testing

- Randomized scheduler with guaranteed probability ($1/(nk^{d-1})$)
- Algorithm:
  1. Assign *n* priority values d, d+1, …, d+n to the *n* threads randomly.
  2. Pick d-1 random priority change points $k_1,\ldots,k_{d-1}$. Each k_i has associated priority value i.
  3. Schedule threads honoring their priorities. When thread reaches i-th change point, change priority of that thread to i.
- Drawback

Typical concurrency bugs and PCT

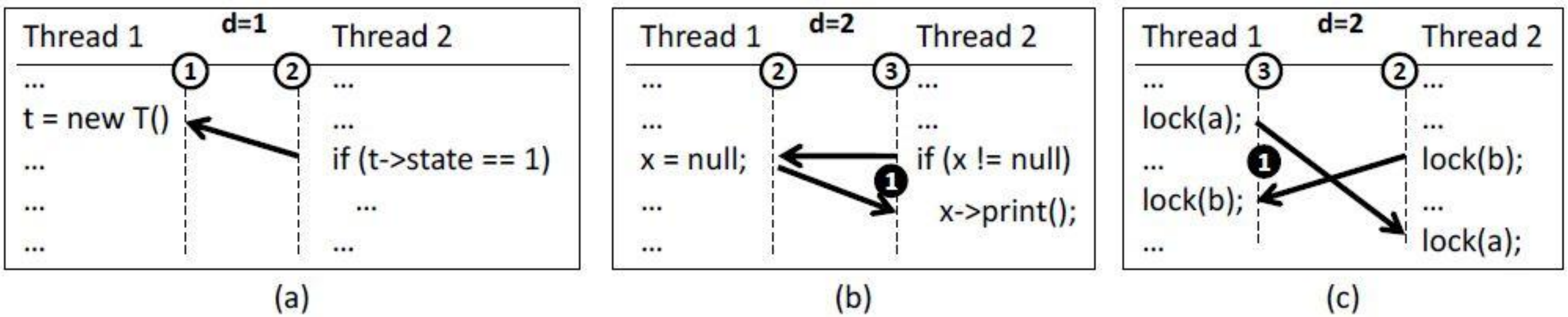


Figure 6. Illustration on how our randomized scheduler finds bugs of depth d, using the examples from Fig. 1. The scheduler assigns random initial thread priorities $\{d, \ldots, d+n-1\}$ (white circles) and randomly places $d-1$ priority change points of values $\{1, \ldots, d-1\}$ (black circles) into the execution. The bug is found if the scheduler happens to make the random choices shown above.

- Image taken from [1]

UNIVERSITÄT SALZBURG

PPCT

- Intuition
  - Two sets of priority threads: High ←→ Low
  - PPCT serlializes execution of lower priority set threads.
  - Number of „Low“ threads is bounded by d.
- Coverage guarantees
- Starvation and Priority Based Scheduling

UNIVERSITÄT SALZBURG

PPCT Algorithm(1)

- Pick a random low priorty thread.
  - At the beginning of the program, pick a thread uniformly at random and assign it a priority d. In addition, insert this thread into the lower priority set L. Insert all other threads into the higher priority set H.

UNIVERSITÄT SALZBURG

PPCT Algorithm(2)

- Pick random priority change points
  - At the beginng of the programm, pick d – 1 priority change points k_1, …, k_{d-1}. Each k_i has an associated priority value of i

UNIVERSITÄT SALZBURG

PPCT Algorithm(3)

- Scheduler choice
  - At each step, the scheduler picks any non-blocked thread in H to schedule. If H is empty or if all threads in H are blocked, the scheduler picks the highest priority thread in L.

UNIVERSITÄT SALZBURG

PPCT Algorithm(4)

- Priority change
  - After each step, increment a step counter. If the step counter matches k_i for some i, change priority of the just executed thread to i and insert it into L.

UNIVERSITÄT SALZBURG

PCT compared to PPCT – Bug Detection

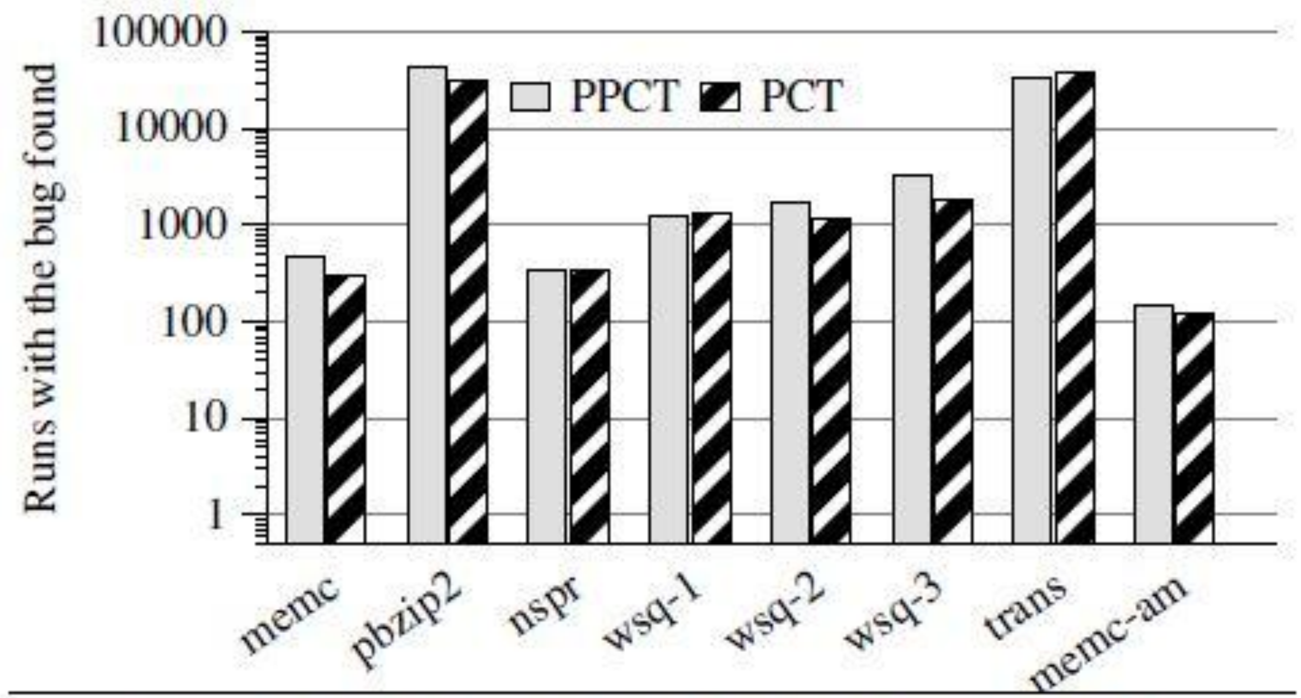


Figure 6. Bug detection of PPCT compared to PCT.

- Image taken from [2]

UNIVERSITÄT SALZBURG

PCT compared to PPCT(2) - Slowdown

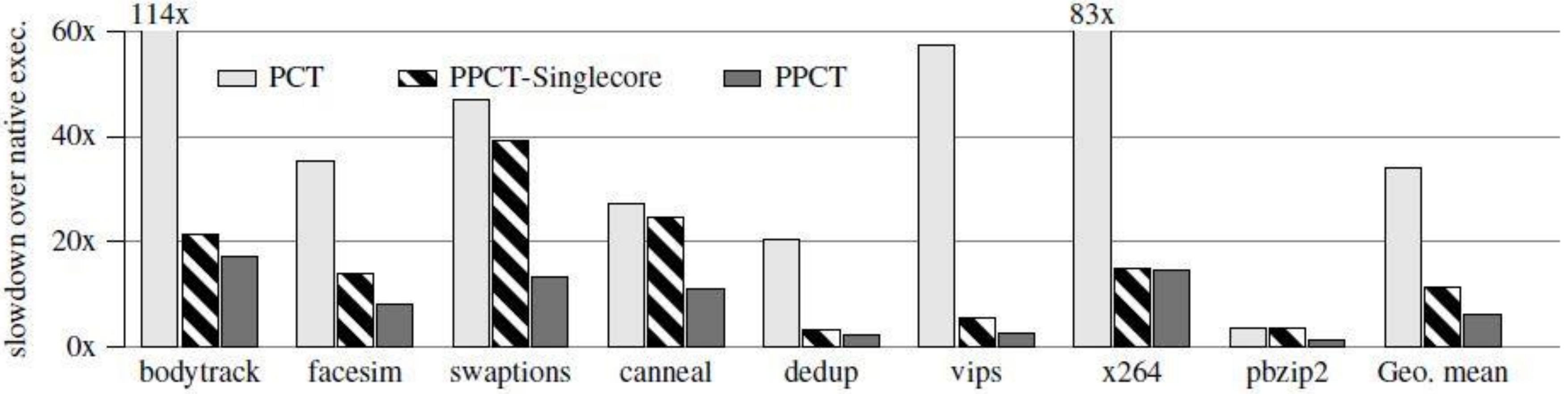


Figure 7. Runtime slowdown of PCT, PPCT on a single core and PPCT when compared to native multithreaded execution.

- Image taken from [2]

UNIVERSITÄT SALZBURG

PCT compared to PPCT(3) - SpeedUp

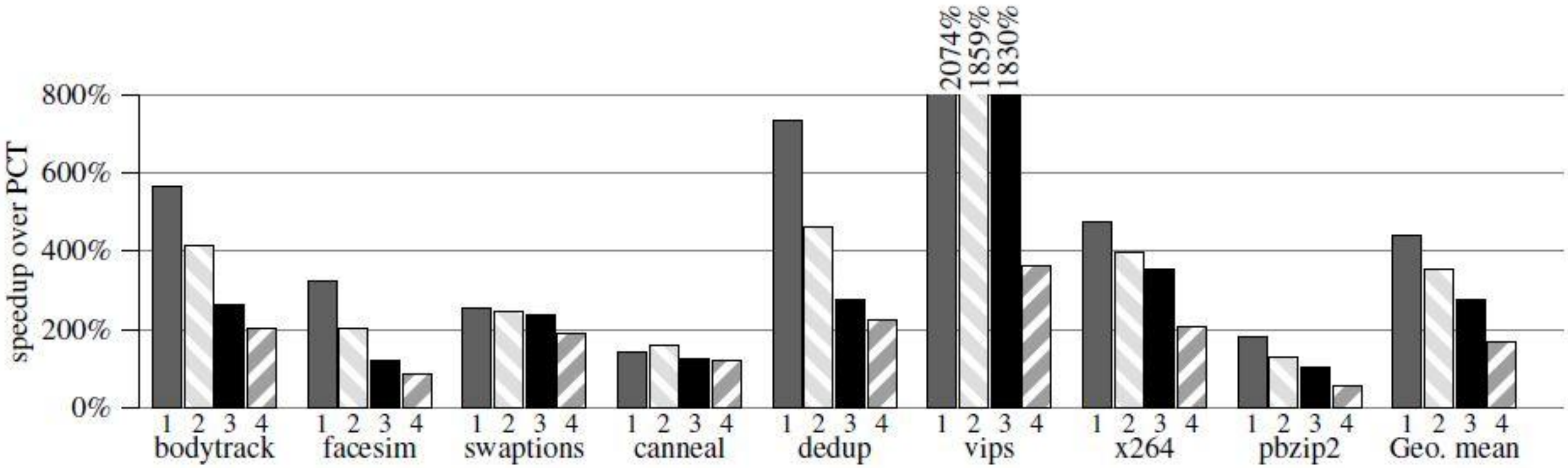


Figure 8. PPCT speedup over PCT that on a multicore machine for explorations with various bug depths. Bug depth of each PPCT exploration is indicated at the bottom of the bar.

- Image taken from [2]

UNIVERSITÄT SALZBURG

Referenzen

- [1] **A Randomized Scheduler with Probabilistic Guarantees of Finding Bugs** by Sebastian Burckhardt, Pravesh Kothari, Madanlal Musuvathi, Santosh Nagarakatte
- [2] **Multicore Acceleration of Prioriy-Based Schedulers for Concurrency Bug Detection** by Santosh Nagarakatte, Sebastian Burckhardt, Milo M. K. Martin, Madanlal Musuvathi
- [3] **Software Testing** Udacity Course by John Regehr and Sean Bennett