Variable and Thread Bounding for Systematic Testing of Multithreaded Programs

Sandeep Bindal, Sorav Bansal
Indian Institute of Technology Delhi

Akash Lal
Microsoft Research India

Motivation

- Concurrency bugs are hard to find
- Difficult to reproduce
- Model Checking too expensive. Search space size for a program with **n** threads and each thread executing **k** instructions $\approx \frac{(\boldsymbol{nk})!}{(\boldsymbol{k}!)^{\boldsymbol{n}}}$

Previous Work (1)

- Musuvathi and Qadeer – Chess[1]:
  - Context Bound 'c' – minimum number of pre-emptive context switches required to uncover the bug.
  - Real world bugs have small c
  - Search Space – $(nk)^c$
  - For $n = 4$, $k = 10{,}000$ and $c = 2$ evaluates to 10^9 - too large
  - Alleviated the search space size problem by considering only synchronization variables.

[1]M. Musuvathi and S. Qadeer. Iterative context bounding for systematic testing of multithreaded programs. In *PLDI '07*.

Previous Work (2)

- Probabilistic Concurrency Testing (PCT)[1]:
  - Bug Depth ‘d’ – minimum number of scheduling constraints required to uncover the bug.
  - Real world bugs have small ‘d’.
  - Search Space – nk^{d-1}
  - Too large for most programs.

[1]S. Burckhardt, P. Kothari, M. Musuvathi, and S. Nagarakatte. A randomized scheduler with probabilistic guarantees of finding bugs. In *ASPLOS ’10*.

Introduction

- Variable Bounding
- Thread Bounding

Variable Bounding

Variable Bounding: Motivation

- Previous studies concluded[1] – 66% of the non-deadlock bugs involve only one variable.
- Remaining fraction of non-deadlock bugs involves only 2-3 variables.
- Data-race involves only one variable.

[1]S. Lu, S. Park, E. Seo, and Y. Zhou. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. In ASPLOS '08.

Variable Bounding: Definition

A **‘v’ variable bug** involve accesses to ‘v’ distinct variables at the minimal set of pre-emptive context switches required to manifest the bug.

Note : Pre-emptive Context Switches, $c \geq v$

Variable Bounding: Examples

a = 0

Thread 1:	Thread 2:
t1 = a;	
	a = 1;
t2 = a;	
assert(t1 == t2);	
	a = 0;

v = 1 and c = 2 bug

Algorithm for v-variable bugs

1) Do lightweight static alias analysis of the test program.

2) For each possible subset of size v variables, repeat steps 3 & 4

3) Instrument the program for the set of v variables.

4) Explore the search space up to context bound ‘c’ of the instrumented program.

Exploration Algorithm

- Explore the search space in order of increasing values of v.
- Real world bugs have small values of v.
- Most bugs will be uncovered at v = 1, 2 and 3.

Search Space Reduction

Assume each thread accesses each variable exactly ‘d’ times and ‘Q’ is the total number of variables.

Search Space at Context Bound ‘c’ = $(ndQ)^c$ …. A

Search Space at ‘v’ and ‘c’ = $\binom{Q}{v}(ndv)^c$ …. B

Assuming v,c << Q, if v < c then B << A

Runtime Improvement

- Instrumentation of tracked variables only.
- Dynamic checking if accessed variable belongs to tracked variables.
- Runtime overhead decreased by 10-100x.
- Enables us to interpose on all variables unlike previous work.

Variable Bounding
Randomized Algorithms

- Applied variable bounding technique in Randomized Algorithms.
- Particularly showed the extension in Probabilistic Concurrency Testing (PCT).
- In general it can be extended to any randomized algorithm.

Probabilistic Concurrency Testing (PCT)[1]

Program with n threads and k instructions:

- Assign n priority values d, d+1,….,d+n-1 randomly to the n threads.
- Pick d-1 priority change points k_1,. . . , k_{d-1} randomly in the range [1,k]. Each k_i has an associated priority of i.
- Schedule the threads by honouring their priorities. When a thread executes k_i change point, change the priority of that thread to i.

[1]S. Burckhardt, P. Kothari, M. Musuvathi, and S. Nagarakatte. A randomized scheduler with probabilistic guarantees of finding bugs. In *ASPLOS '10*.

PCT with Variable Bounding Algorithm

Program with n threads and k instructions:

- Choose a set of ‘v’ variables randomly.
- Assign n priority values d, d+1,….,d+n-1 randomly to the n threads.
- Pick d-1 priority change points $k_1, \ldots, k_{d-1}$ randomly in the range [1,k], such that the variables accessed at these priority change points belongs to set of randomly selected variables.
- Schedule the threads by honouring their priorities. When a thread reaches the i-th change point, change the priority of that thread to i.

Probabilistic Guarantees

$$\text{Probability with PCT} = \frac{1}{n\,k^{d-1}}$$

$$\text{Probability with PCT-VB} = \frac{1}{\binom{Q}{v}} \frac{1}{n\,k'^{d-1}}$$

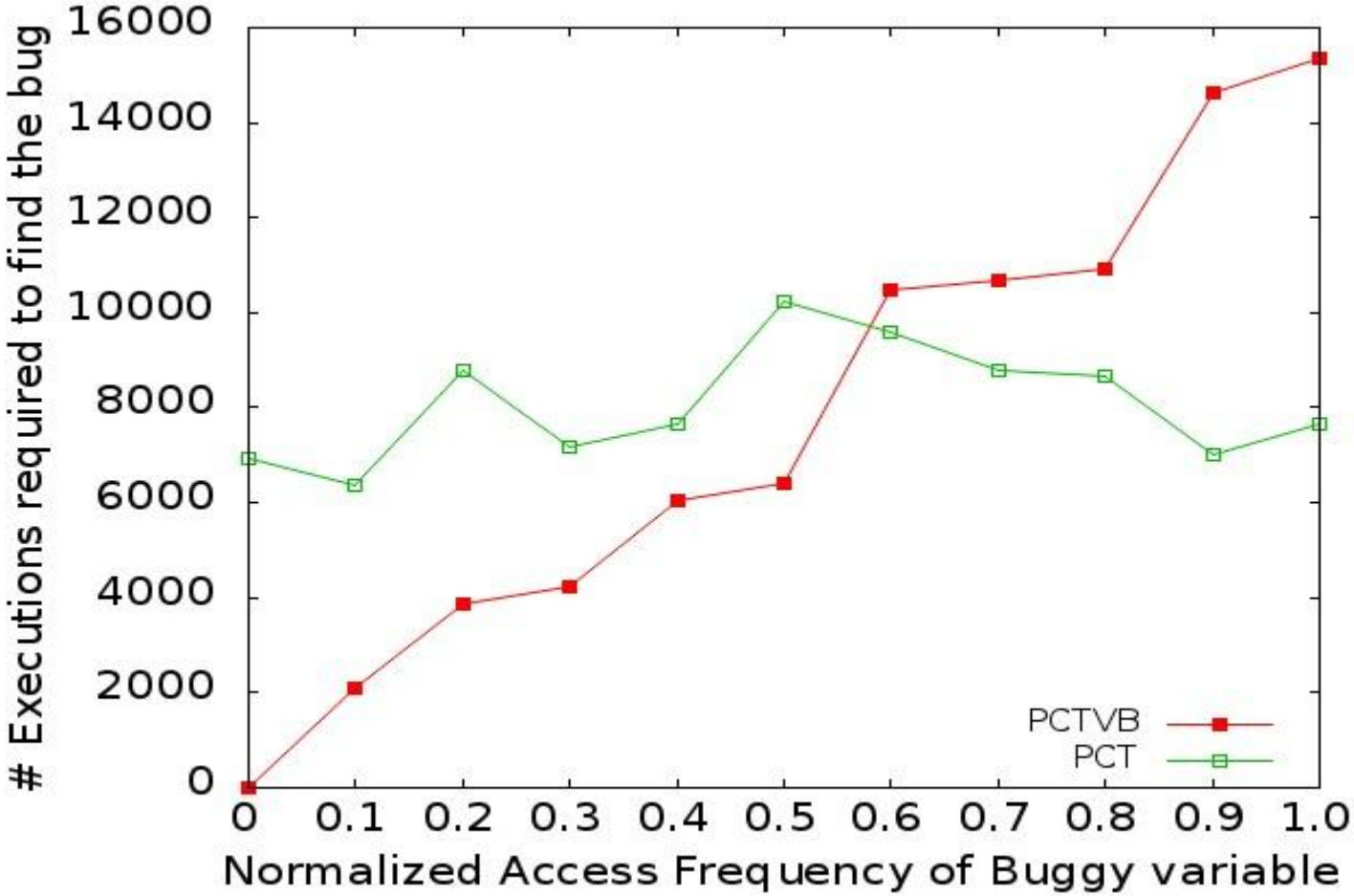
(k' - maximum number of accesses of selected variables)

Assuming v << Q

if v < d – 1 → **Pr-PCTVB >> Pr-PCT**

if v == d-1 → **Pr-PCTVB ≈ Pr-PCT**

Improvement in Probability for corner variables



Thread Bounding

Thread Bounding

- Most concurrency bugs involve interaction between small number of threads (mostly 2).
- Data-race involve interactions between only two threads.

Thread Bounding: Definition

A t-thread bug requires ordering constraints between at least ‘t’ distinct threads to manifest the bug.

By definition: $t \geq 2$

Thread Bounding: Examples

a = 0;

Thread 1:	Thread 2:	Thread 3:
a++;		
		a++;
	assert(a != 2);	

c = 0, v = 0 and t = 3 bug

Thread Bounding

A program is t-thread-bound bug free if all relative orders of t-thread-bound bugs are explored.

Thread Bounding: Search Space Size

- c = 0 and t = 2
- c = 0 and any general ‘t’
- any general ‘c’ and ‘t’

Thread Bounding: Search Space Size
c = 0 and t = 2

- Only two executions are required
  - One executing threads T_1, T_2, …. T_n in ascending order $\{T_1, T_2, \ldots, T_n\}$
  - Other executing threads in descending order $\{T_n, T_{n-1}, \ldots T_1\}$

- Cover both orderings of all 2-sized subsets of all threads.

- For every T_i, T_j pair both T_i-->T_j and T_j-->T_i enumerated

Thread Bounding: Search Space Size

c = 0 and any general ‘t’

- Need to cover all orderings of t-sized subsets of n threads.
- Showed that $(t+1)!\ (\log(nt) + \log(\frac{1}{\varepsilon}))$ ordering required to explore all t-thread bugs with probability at least $(1- \varepsilon)$.
- Therefore, $O((t+1)!\log(n))$ schedules are sufficient to cover all t-thread bugs at c = 0 with a high probability.

Thread Bounding: Search Space Size

c = 0 and any general ‘t’

- Example: n = 600
  - t = 3 → 70 orderings enough to uncover bug with probability > 99%
  - t = 4 → 360
  - t = 5 → 2000

Thread Bounding: Search Space Size

any general ‘c’ & ‘t’

- For context bound c, consider a pre-empted thread as two distinct threads. Total number of threads = n + c.
- Explore all ordering of (t+c)-sized subsets of n+c threads.
- O((t+c+1)!log(n+c)) schedules are sufficient to capture all c-context & t-thread bugs.

Thread Bounding: Algorithm

1) Generate a random ordering of (n+c) threads.

2) Schedule the threads according to the random priority ordering.

3) On the ith pre-emptive context switch change current thread id to n+ i and update its priority corresponding to n+i.

4) Repeat 1-3 for (t+c+1)! log(n+c) times.

Variable & Thread Bounding

- Independent techniques.
- Can be used together to provide cumulative improvements.

Implementation

- Implemented Variable and Thread bounding in Java tool – Rankchecker.
- Static Alias Analysis using BDDs.
- Rankchecker implements scheduler which dictates thread interleavings.

Experimental Results

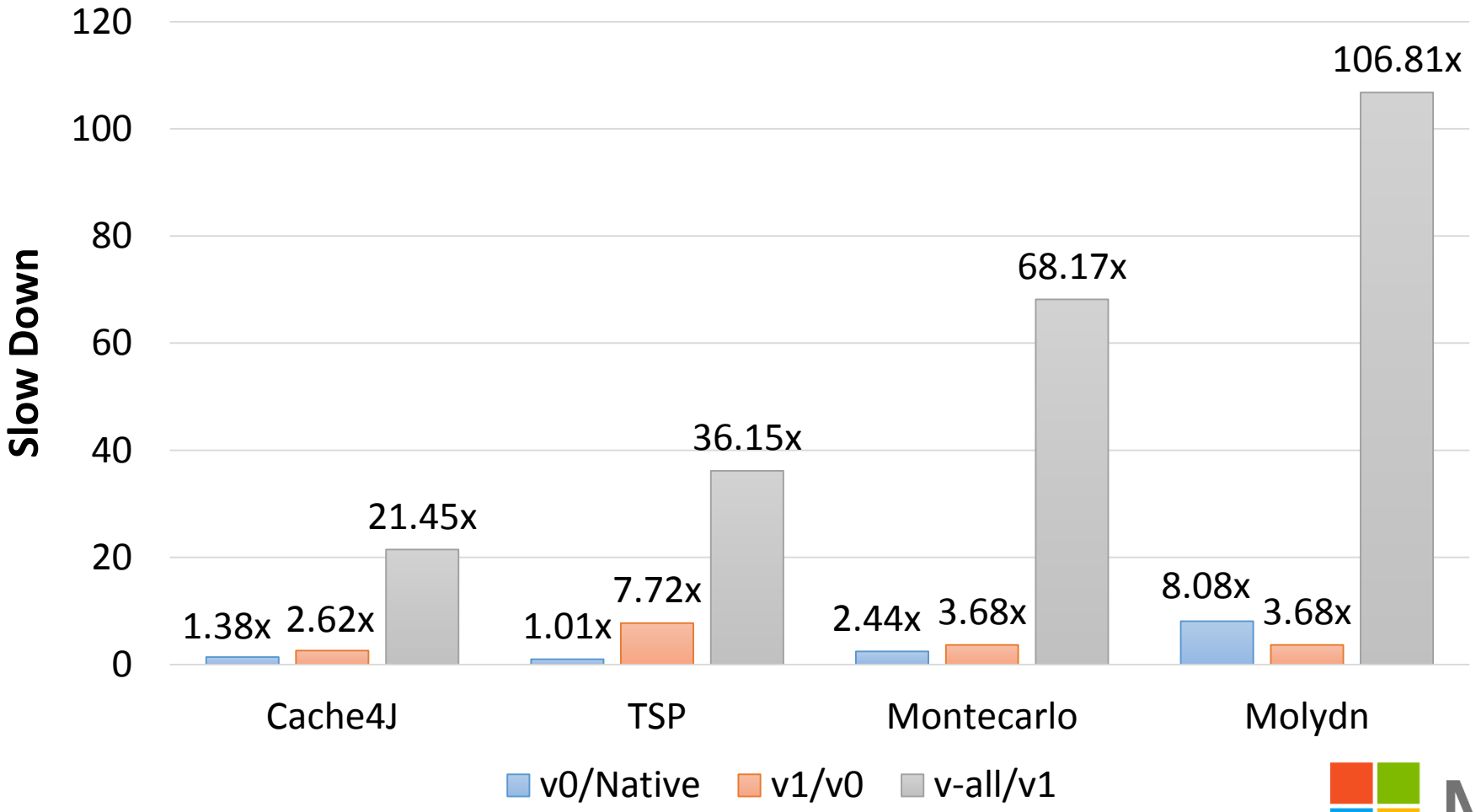
Experimental Results

- Typical values of ‘v’ and ‘t’ in common concurrency bugs?
- Runtime Improvement due to variable bounding?
- Improvements in number of executions using systematic search strategy and randomized search strategy, due to v-t bounding?

Experimental Results

Benchmark	Num of Threads	Schedules Explored	(c,v,t)
ConTest Benchmarks			
MergeSort	100	651	(1,1,2)
PingPong	18	234	(1,1,2)
AirLine Tickets	99	2	(0,0,2)
AllocationVector	3	113	(2,1,2)
Manager	6	33	(1,1,2)
Java library JDK 1.4.2			
HashSet	200	813	(1,1,2)
TreeSet	200	813	(1,1,2)

Experiments: Runtime Improvements



Experiments: Variable Bounding in Delay Bounded-CHESS[1]

RegionOwnership	**(C#)**		
	BugFound?	#Executions	Time(sec)
No VB	Yes	132507	6900
v=2	Yes	47248	1225
v=1	No	30437	581.0

[1]M. Emmi, S. Qadeer, and Z. Rakamaric. Delay-bounded scheduling, In *POPL, pages 411-422, 2011*.

Experiments
Randomized Search Strategy

MonteCarlo Benchmark

Bug Type	With v-t bounding		Without v-t bounding	
(c ,v ,t)	Avg # Executions	Time(sec)	Avg # Executions	Time(sec)
(0,0,2)	3.1	10.9	2.7	768.9
(0,0,3)	3.7	12.1	3.5	1010.7
(1,1,2)	1636.2	6409.9	-	TimedOut
(1,1,3)	4889	20371.2	-	TimedOut
(2,1,2)	28121	112950	-	TimedOut

Conclusion

- Most bugs can be manifest at small values of v and t.
- Variable and thread bounding significantly reduces the search space.
- Due to variable bounding, can interpose on all variable accesses not just synchronization variables.
- Showed improvements over existing tools (CHESS and PCT).

Thank You!

Questions?