DataRaceBench: A Benchmark Suite for Systematic Evaluation of Data Race Detection Tools

Chunhua “Leo” Liao, Pei-Hung Lin, Joshua Asplund, Markus Schordan and Ian Karlin

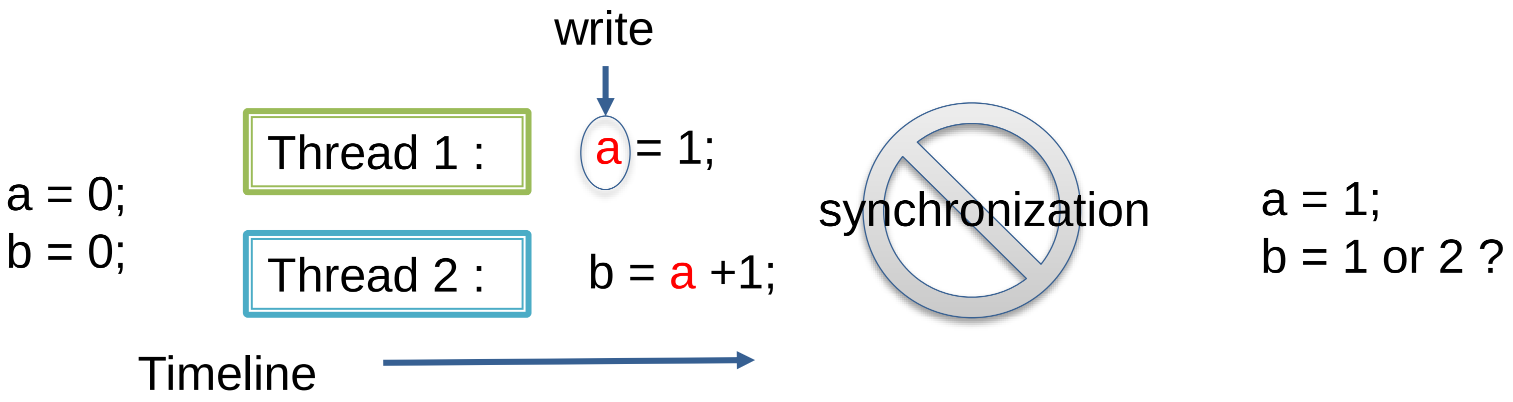
SC17, Denver, CO

Nov. 14th, 2017

LLNL-PRES-739610

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.

What is a data race?



Data race bugs:

- Computation may give different results from run to run depending on memory access order.
- Threat to correctness of all multithreaded applications, including HPC applications

Detecting and eliminating data races can be lifesaving

Radiation therapy machine accidents in 1980s

Northeast blackout in 2003

Self-driving cars?

Motivation

Data Race Detection Tools	Benchmarks
Archer '16	Kernels (OmpSCR), Real/proxy apps (AMG2013, HYDRA)
PolyOMP '16	Kernels (OmpSCR), Perf. bench.(PolyBench-ACC)
Versatile On-the-fly Race Detection'14	Perf. bench. (ECPP, NPB)
OpenMP Analysis Toolkit (OAT)'13	Perf. bench. (NPB)
RaceMob '13	Real apps. (Apache httpd, SQLite, Memcached)
ompVerify '11	Kernels (Stencil, matrix transpose, sort)
Intel Thread Checker'03	Kernels (Histogram)

Problem: no dedicated benchmark suites to evaluate data race detection tools

Solution: a dedicated data race detection benchmark suite

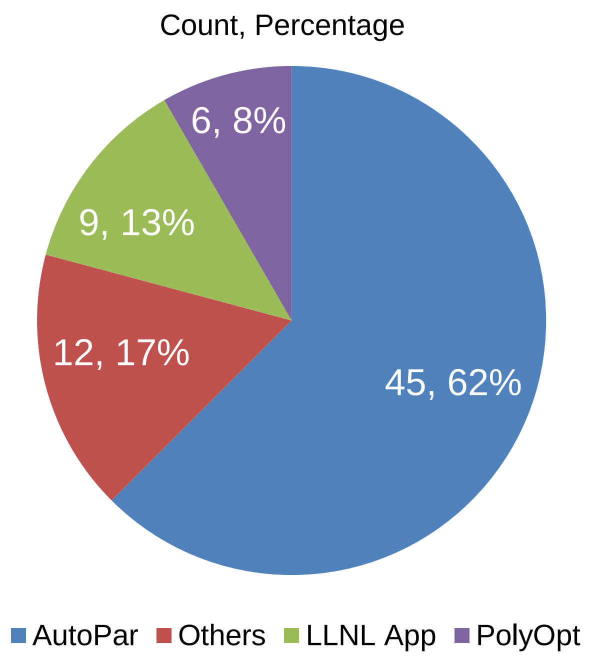
- Focus on OpenMP
- Capture data race patterns
- Generate quantitative metrics
- Discover strengths and limitations of data race detection tools

Design criteria and solutions

Criteria	Our Solutions
Representative	Multiple sources to capture typical OpenMP patterns
Scalable	Allow different data sizes and thread counts
General	Support both static and dynamic tools
Accessible	BSD license, github.com/LLNL/dataracebench
Extensible	Self-contained programs
Easy to use	Automated scripts for execution and report generation
Quantitative	Generate a range of standard metrics in reports
Correct	Use compilers and correctness tools, and manual inspection

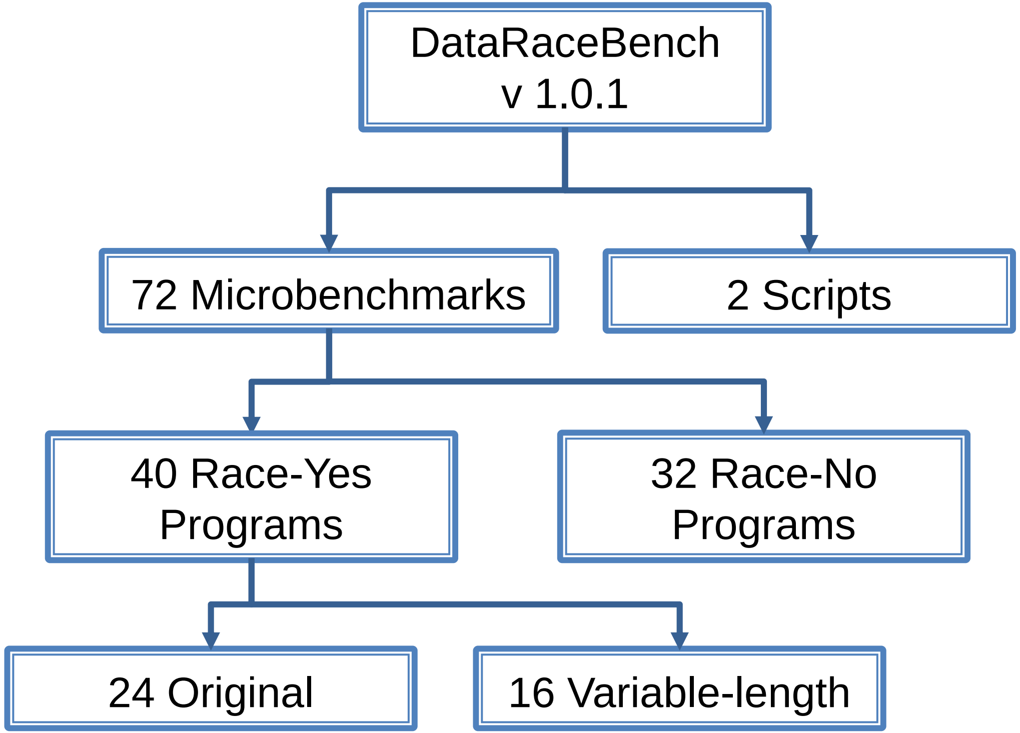
Multiple sources to ensure representativeness of microbenchmarks

- OpenMP regression tests
  — from an automatic parallelization tool, AutoPar[1]
- Optimized OpenMP programs
  — generated by PolyOpt[2]
- Historic data races found in real LLNL applications
- Others: additional OpenMP data race patterns from literature



1. http://rosecompiler.org/ROSE_HTML_Reference/auto_par.html
2. http://web.cs.ucla.edu/~pouchet/software/polyopt/

Content of DataRaceBench



- When applicable, each race-yes program
  - has at most a single pair of source locations causing runtime data races
  - pairs up with a race-no program

Data race patterns: property labels & number of microbenchmarks for each label

Property labels for race-yes set	#	Property labels for race-no set	#
Y1: Unresolvable dependences	23	N1: Embarrassingly parallel	7
Y2: Missing data sharing clauses	6	N2: Use of data sharing clauses	9
Y3: Missing synchronization	4	N3: Use of explicit synchronization	2
Y4: SIMD data races	2	N4: Use of SIMD directives	4
Y5: Accelerator data races	1	N5: Use of accelerator directives	1
Y6: Undefined behaviors	2	N6: Use of special language features	5
Y7: Numerical kernel data races	4	N7: Numerical kernels	9
Total #	42		37

Examples related to data sharing clauses

```
1.    …
2.    int i,x;
3.    #pragma omp parallel for
4.     for (i=0;i<100;i++)
5.     {   x=i;  }
6.     printf("x=%d",x);
7.    …
```

one data race pair

x@5 vs. x@5

Y2: Missing data sharing clauses

lastprivatemissing-orig-yes.c

```
1.    …
2.    int i,x;
3.    #pragma omp parallel for lastprivate (x)
4.    for (i=0;i<100;i++)
5.     {   x=i; }
6.    printf("x=%d",x);
7.    …
```

N2: Use of data sharing clauses

lastprivate-orig-no.c

Example related to synchronization

```
1.      #pragma omp parallel shared(b, error)
2.      {
3.      #pragma omp for nowait
4.      for(i = 0; i < len; i++)
5.        a[i] = b + a[i]*5;

6.       #pragma omp single
7.         error = a[9] + 1;
8.      }
```

nowait-orig-yes.c

Y3: Missing synchronization

Data race pair:
a[i]@5 vs. a[9]@7

Example originated from "out-of-bounds" accesses

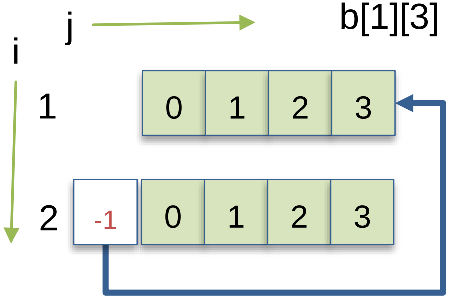
```
1.    double b [n] [m] ;
2.    #pragma omp parallel for private ( j )
3.    for ( i = 1 ; i <n ; i ++)
4.      for ( j =0 ; j <m; j ++)
5.         b [ i ] [ j ]= b[ i ] [ j −1 ] ;
```

outofbounds-orig-yes.c

Assuming n=4, m=4
multidimensional arrays
use row-major order storage

Data race pair:
b[i][j]@5 vs. b[i][j-1]@5

i=1, j=3
b[1][3]



i=2, j=0
b[2][0 − 1]

Example related to numerical kernel data races

```
1.  int indexSet[180] = {
2.  521, 523, 525, 527, 529, 533,
3.  547, 549, 551, 553, 555, 557,...
4.   };
5.  double * xa1, *xa2; ...
6.  xa2 = xa1 + 12 ;
7.  #pragma omp parallel for
8.  for(int i=0; i< 180; ++i)
9.  {
10.   int idx=indexSet[i];
11.   xa1[idx]+=1.0;
12.   xa2[idx]+=3.0;
13. }
```

indirectaccess2-orig-yes.c
Data race pair:
xa1[idx]@11 vs. xa2[idx]@12

iteration i	**0**	**1**	**…**	**5**
indexSet[i]	521	523	…	533
xa1[indexSet[i]]	base+521		…	base+533
xa2[indexSet[i]]	base+12+521		…	

Test sensitivity to thread count & loop scheduling

Loop iterations

0	1	2	3	4	5	…	177	178	179

Loop schedule
with 2 threads

0	…	5	…	89

thread 1/2

90	91	..	178	179

thread 2/2

Loop schedule 2
with 36 threads

180/36=5

0	…	4

thread 1/36

5	…	9

thread 1/36

…

Evaluation

- DataRaceBench
  - https://github.com/LLNL/dataracebench v1.0.1
- Tools
  - Pthreads-based: Helgrind, ThreadSanitizer
  - OpenMP-aware: Archer, Intel Inspector
- Hardware
  - Quartz cluster@LLNL supporting 72 threads per node
- Execution configuration*: (4+1) x 5320
  - Tools x microbenchmarks x OpenMP threads x Array Sizes x Repeats

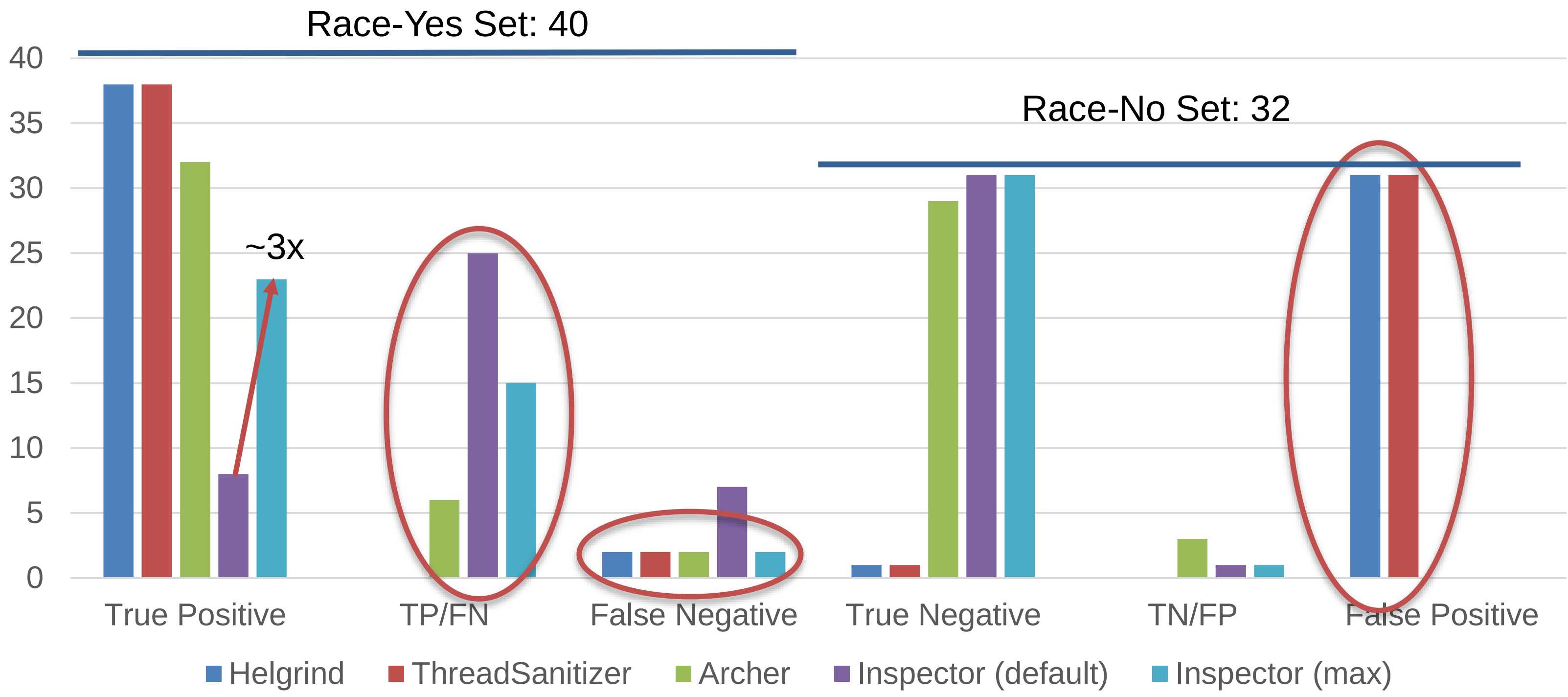
* Two configurations of Intel Inspector were tested: default vs. maximum resources

Results : automatically generated report (partial)

Race-Yes	Helgrind		ThreadSanitizer		Archer		Intel Inspector (max resource)	
ID	Race #	Type	Race#	Type	Race#	Type	Race#	Type
1	12-15	TP	3-188	TP	2-161	TP	1-1	TP
5	13-15	TP	4-14	TP	1-1	TP	0-1	TP FN
6	10-15	TP	3-14	TP	0-1	TP FN	0-1	TP FN
25	0-0	FN	0-0	FN	0-0	FN	0-0	FN

Term	Meaning in our context
True Positive (TP)	Detecting data races in a race-yes program
False Positive (FP)	Detecting data races in a race-no program
True Negative (TN)	Not detecting data races in a race-no program
False Negative (FN)	Not detecting data races in a race-yes program

Results: positive and negative counts



Results: standard metrics

Tool	Precision		Recall		Accuracy	
	min	max	min	max	min	max
Helgrind	0.551	0.551	0.950	0.950	0.542	0.542
ThreadSanitizer	0.551	0.551	0.950	0.950	0.542	0.542
Archer	0.914	1.000	0.800	0.950	0.847	0.972
Intel Inspector(default)	0.889	1.000	0.200	0.825	0.542	0.903
Intel Inspector(max)	0.958	1.000	0.575	0.950	0.750	0.972

Metric	Formula
Precision	Confidence of true positive P = TP/(TP + FP)
Recall	Completeness of true positive R = TP/(TP + FN)
Accuracy	Chance of having a correct report A = (TP +TN)/(TP + FP +TN + FN)

Conclusion

- DataRaceBench: a dedicated OpenMP benchmark suite to evaluate data race detection tools
  - Enable systematic and quantitative evaluations
  - Very positive evaluation results
- Lessons about execution settings
  - Configurations of dynamic tools matter: Intel default vs. max resources
  - Multiple runs: necessary to increase probability of finding data races
  - Sensitive to the number of threads and scheduling policies
- Findings about results
  - Precision/Accuracy: Archer and Intel Inspector win over Helgrind and ThreadSanitizer due to OpenMP awareness
  - User friendliness: Only Intel inspector consolidates multiple data race instances into one single pair of source locations
  - SIMD loops with data races: compilers do no generate SIMD instructions for our race-yes SIMD benchmarks

Q&A

- https://github.com/LLNL/dataracebench

Lawrence Livermore National Laboratory

Related work

- Benchmarks
  - Performance benchmarks:
    - SPEC, LINPAC
    - SPECOMP, NAS Parallel Benchmark, Rodinia, OMPSRC
  - Correctness benchmarks
    - BugBench: only two data race tests
    - JAVA: Modified Java Grande, DaCaPo
- Tools/Algorithm evaluation
  - Single tool: Intel Thread Checker: benchmark suite not released
  - Multi-tool: [Alowibdi2013]RaceFuzzer, RacerAJ, JCHORD, Race Condition Checker, Java RaceFinder
  - Multi-algorithm: [Yu2017]Dynamic detection methods including FastTrack, Acculock, Multilock-HB, SimpleLock+, and Casually Precedes Detection