LLOV: A Fast Static Data-Race Checker for OpenMP Programs

Utpal Bora

PhD Student
Computer Science and Engineering
IIT Hyderabad, India

February 23, 2020

Table of Contents

Data race in Parallel programs

Definition (Data Race)

An execution of a concurrent program is said to have a *data race* when two different threads access the same memory location,

- these accesses are not protected by a mutual exclusion mechanism
- the order of the two accesses is non-deterministic
- one of these accesses is a write

Common race conditions in OpenMP programs

- Missing data sharing clauses

```
#pragma omp parallel for
    private (temp,i,j)
  for (i = 0; i < len; i++)
    for (j = 0; j < len; j++)
    {
      temp = u[i][j];
      sum = sum + temp * temp;

    }
```

DRB021: OpenMP Worksharing construct with data race

Common race conditions in OpenMP programs

- Missing data sharing clauses
- Loop carried dependences

```
for (i=0;i<n;i++) {
#pragma omp parallel for
  for (j=1;j<m;j++) {
    b[i][j]=b[i][j-1];
  }
}
```

DRB038: Example with Loop Carried Dependence

Common race conditions in OpenMP programs

- Missing data sharing clauses
- Loop carried dependences
- SIMD races

```
#pragma omp simd
for (int i=0; i<len-1; i++){
  a[i+1] = a[i] + b[i];
}
```

DRB024: Example with SIMD data race

Common race conditions in OpenMP programs

- Missing data sharing clauses
- Loop carried dependences
- SIMD races
- Synchronization issues

```
#pragma omp parallel shared(b,
      error) {
#pragma omp for nowait
    for(i = 0; i < len; i++)
      a[i] = b + a[i]*5;
#pragma omp single
    error = a[9] + 1;
  }
```

DRB013: Example with data race due to improper synchronization

Common race conditions in OpenMP programs

- Missing data sharing clauses
- Loop carried dependences
- SIMD races
- Synchronization issues
- Control flow dependent on number of threads

```
#pragma omp parallel
  if (omp_get_thread_num() % 2
      == 0) {
    Flag = true;
  }
```

Control flow dependent on number of threads

Race Detection Tools

Table: OpenMP Race Detection Tools: A Short Survey

Tools	Infrastructure	Analysis Type
HELGRIND [Vp07b]	Valgrind	Dynamic
VALGRIND DRD [Vp07a]	Valgrind	Dynamic
TSAN [SI09]	LLVM/GCC	Dynamic
ARCHER [AGR+16]	LLVM	Hybrid
SWORD [AGR+18]	LLVM	Dynamic
ROMP [GMC18]	Dyninst	Dynamic
POLYOMP [CSS15]	ROSE	Static
DRACO [YSL+18]	ROSE	Static
OMPVERIFY [BYR+11]	AlphaZ	Static

Race Detection Tools

Table: OpenMP Race Detection Tools: A Short Survey

Tools	Infrastructure	Analysis Type
HELGRIND [Vp07b]	Valgrind	Dynamic
VALGRIND DRD [Vp07a]	Valgrind	Dynamic
TSAN [SI09]	LLVM/GCC	Dynamic
ARCHER [AGR+16]	LLVM	Hybrid
SWORD [AGR+18]	LLVM	Dynamic
ROMP [GMC18]	Dyninst	Dynamic
POLYOMP [CSS15]	ROSE	Static
DRACO [YSL+18]	ROSE	Static
OMPVERIFY [BYR+11]	AlphaZ	Static

There is still need for a static OpenMP data race checker in LLVM.

Advantage of Static tools over Dynamic tools

Static tools have the following advantages over dynamic tools:

- Can detect races in SIMD constructs

Advantage of Static tools over Dynamic tools

Static tools have the following advantages over dynamic tools:

- Can detect races in SIMD constructs
- Are independent of the runtime thread schedule

Advantage of Static tools over Dynamic tools

Static tools have the following advantages over dynamic tools:

- Can detect races in SIMD constructs
- Are independent of the runtime thread schedule
- Are independent of the input size

Advantage of Static tools over Dynamic tools

Static tools have the following advantages over dynamic tools:

- Can detect races in SIMD constructs
- Are independent of the runtime thread schedule
- Are independent of the input size
- Are independent of the number of threads

Advantage of Static tools over Dynamic tools

Static tools have the following advantages over dynamic tools:

- Can detect races in SIMD constructs
- Are independent of the runtime thread schedule
- Are independent of the input size
- Are independent of the number of threads

Advantage of Static tools over Dynamic tools

Static tools have the following advantages over dynamic tools:

- Can detect races in SIMD constructs
- Are independent of the runtime thread schedule
- Are independent of the input size
- Are independent of the number of threads

LLOV is an attempt to bridge this gap and move towards a fast, language agnostic, robust, static OpenMP data race checker in LLVM.

Table of Contents

LLOV Overview

LLOV is a language agnostic, static OpenMP data race checker in the LLVM compiler framework.

LLOV Overview

LLOV is a language agnostic, static OpenMP data race checker in the LLVM compiler framework. LLOV is

- based on Intermediate representation of LLVM (LLVM-IR)

LLOV Overview

LLOV is a language agnostic, static OpenMP data race checker in the LLVM compiler framework. LLOV is

- based on Intermediate representation of LLVM (LLVM-IR)
- uses Polyhedral framework, Polly, of LLVM

LLOV Overview

LLOV is a language agnostic, static OpenMP data race checker in the LLVM compiler framework. LLOV is

- based on Intermediate representation of LLVM (LLVM-IR)
- uses Polyhedral framework, Polly, of LLVM
- can handle FORTRAN as well as C/C++

LLOV Overview

LLOV is a language agnostic, static OpenMP data race checker in the LLVM compiler framework. LLOV is

- based on Intermediate representation of LLVM (LLVM-IR)
- uses Polyhedral framework, Polly, of LLVM
- can handle FORTRAN as well as C/C++
- can detect that a program is race free

LLOV Overview

LLOV is a language agnostic, static OpenMP data race checker in the LLVM compiler framework. LLOV is

- based on Intermediate representation of LLVM (LLVM-IR)
- uses Polyhedral framework, Polly, of LLVM
- can handle FORTRAN as well as C/C++
- can detect that a program is race free
- has all the advantages of a static data-race checker

LLOV Overview

LLOV is a language agnostic, static OpenMP data race checker in the LLVM compiler framework. LLOV is

- based on Intermediate representation of LLVM (LLVM-IR)
- uses Polyhedral framework, Polly, of LLVM
- can handle FORTRAN as well as C/C++
- can detect that a program is race free
- has all the advantages of a static data-race checker
- can be extended for approximate dependences (like LAI of LLVM)

LLOV Overview

LLOV is a language agnostic, static OpenMP data race checker in the LLVM compiler framework. LLOV is

- based on Intermediate representation of LLVM (LLVM-IR)
- uses Polyhedral framework, Polly, of LLVM
- can handle FORTRAN as well as C/C++
- can detect that a program is race free
- has all the advantages of a static data-race checker
- can be extended for approximate dependences (like LAI of LLVM)
- has provision for handling entire OpenMP pragmas

LLOV Architecture

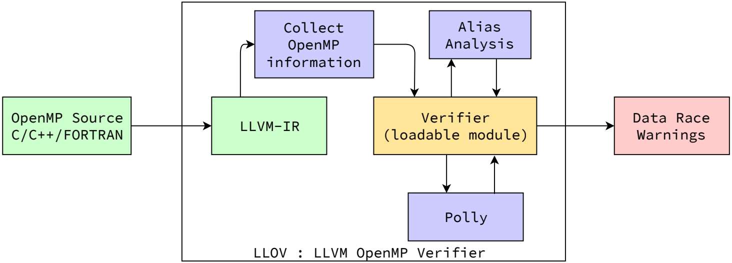

Figure: Flow Diagram of LLVM OpenMP Verifier (LLOV)

Methodology (with Example)

```
for (i=0;i<10;i++) {
#pragma omp parallel for
  for (j=1;j<10;j++) {
    b[i][j]=b[i][j-1];
  }
}
```

Example with Loop Carried Dependence

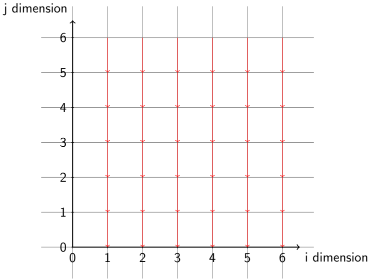

Figure: Dependence Polyhedra

Methodology (with Example)

```
for (i=0;i<10;i++) {
#pragma omp parallel for
  for (j=1;j<10;j++) {
    b[i][j]=b[i][j-1];
  }
}
```

Listing 1: Example with Loop Carried Dependence

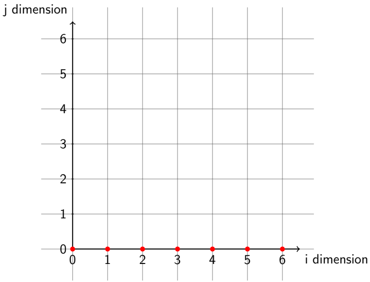

Figure: Projection of the Dependence Polyhedra on i-dimension

Zero magnitude of the projections on a dimension signifies that the dimension is parallel.

Methodology (with Example)

```
for (i=0;i<10;i++) {
#pragma omp parallel for
  for (j=1;j<10;j++) {
    b[i][j]=b[i][j-1];
  }
}
```

Listing 2: Example with Loop Carried Dependence

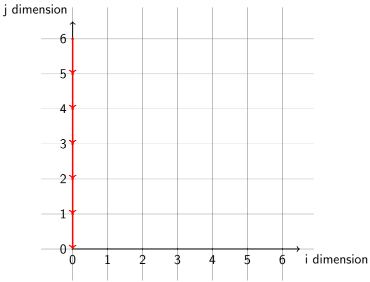

Figure: Projection of the Dependence Polyhedra on j-dimension

Non-zero magnitude of the projections on a dimension signifies that the dimension is not parallel.

Table of Contents

Results: Experimental Setup

Benchmarks:

- DataRaceBench C/C++ v1.2 [LLA+18, LLSK18]
- OmpSCR v2.0 [Dor04, DRd05]
- DataRaceBench FORTRAN [KSB19]

System Specifications:

System: Two Intel Xeon E5-2697 v4 @ 2.30GHz processors
OS: 64 bit Ubuntu 18.04.2 LTS server
Kernel: Linux kernel version 4.15.0-48-generic
Threads: 72 (2 x 36) hardware threads
Memory: 128GB
OpenMP library: LLVM OpenMP runtime v5.0.1 (libomp5)

Results: Other Race Detection Tools

Table: Race detection tools with the version numbers used for comparison

Tools	**Source**	**Version / Commit**
HELGRIND [Vp07b]	Valgrind	3.13.0
VALGRIND DRD [Vp07a]	Valgrind	3.13.0
TSAN-LLVM [SI09]	LLVM	6.0.1
ARCHER [AGR+16]	git master branch	fc17353
SWORD [AGR+18]	git master branch	7a08f3c
ROMP [GMC18]	git master branch	6a0ad6d

Results: DataRaceBench v1.2 comparison

Table: Maximum number of Races reported by different tools in DataRaceBench 1.2

Tools	Race: Yes		Race: No		Coverage/116
	TP	FN	TN	FP	
HELGRIND	56	3	2	55	116
VALGRIND DRD	56	3	26	31	116
TSAN-LLVM	**57**	2	2	55	116
ARCHER	56	3	2	55	116
SWORD	47	4	24	4	79
LLOV	45	3	**28**	9	85

Results: DataRaceBench v1.2 comparison

Table: Maximum number of Races reported by different tools in DataRaceBench 1.2

Tools	Race: Yes		Race: No		Coverage/116
	TP	FN	TN	FP	
Helgrind	56	3	2	55	116
Valgrind DRD	56	3	26	31	116
TSan-LLVM	**57**	2	2	55	116
Archer	56	3	2	55	116
SWORD	47	4	24	4	79
LLOV	45	3	**28**	9	85

Table: Maximum number of Races reported by different tools in common 66 kernels of DataRaceBench 1.2

Tools	Race: Yes		Race: No		Coverage/116
	TP	FN	TN	FP	
Helgrind	46	1	2	17	66
Valgrind DRD	46	1	13	6	66
TSan-LLVM	46	1	2	17	66
Archer	46	1	2	17	66
SWORD	46	1	18	1	66
LLOV	44	3	16	3	66

Results: DataRaceBench v1.2 statistics

Table: Precision, Recall and Accuracy of the tools on DataRaceBench 1.2

Tools	Precision	Recall	Accuracy	F1 Score	Diagnostic odds ratio
HELGRIND	0.50	0.95	0.50	0.66	0.68
VALGRIND DRD	0.64	0.95	0.71	0.77	15.66
TSAN-LLVM	0.51	**0.97**	0.51	0.67	1.04
ARCHER	0.50	0.95	0.50	0.66	0.68
SWORD	**0.92**	0.92	**0.90**	**0.92**	**70.50**
LLOV	0.83	0.94	0.86	0.88	46.67

Results: DataRaceBench v1.2 statistics

Table: Precision, Recall and Accuracy of the tools on DataRaceBench 1.2

Tools	Precision	Recall	Accuracy	F1 Score	Diagnostic odds ratio
Helgrind	0.50	0.95	0.50	0.66	0.68
Valgrind DRD	0.64	0.95	0.71	0.77	15.66
TSan-LLVM	0.51	**0.97**	0.51	0.67	1.04
Archer	0.50	0.95	0.50	0.66	0.68
SWORD	**0.92**	0.92	**0.90**	**0.92**	**70.50**
LLOV	0.83	0.94	0.86	0.88	46.67

Table: Precision, Recall and Accuracy of the tools on common 66 kernels of DataRaceBench 1.2

Tools	Precision	Recall	Accuracy	F1 Score	Diagnostic odds ratio
Helgrind	0.73	**0.98**	0.73	0.84	5.41
Valgrind DRD	0.88	**0.98**	0.89	0.93	99.67
TSan-LLVM	0.73	**0.98**	0.73	0.84	5.41
Archer	0.73	**0.98**	0.73	0.84	5.41
SWORD	**0.98**	**0.98**	**0.97**	**0.98**	**828.00**
LLOV	0.94	0.94	0.91	0.94	78.22

Results: DataRaceBench v1.2 runtime

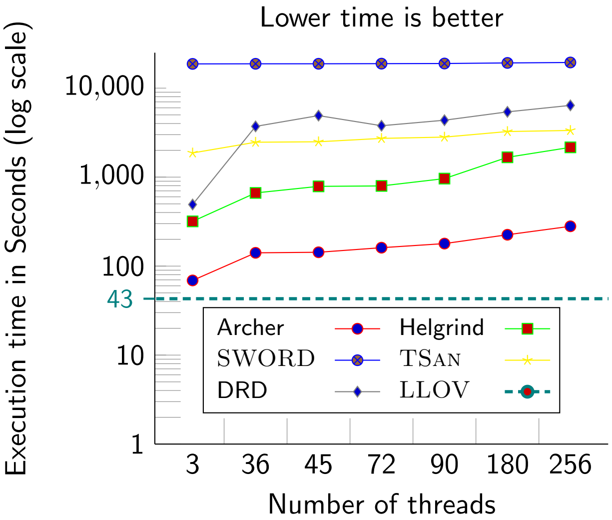

Figure: DataRaceBench v1.2 total time taken by different tools for all 116 kernels on logarithmic scale

Results: DataRaceBench v1.2 runtime

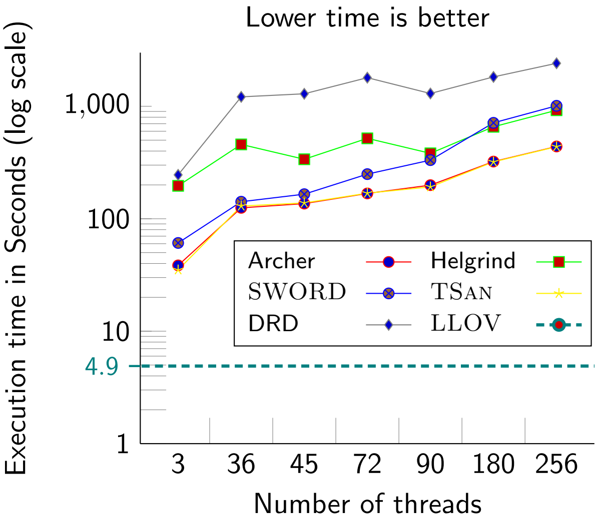

Figure: DataRaceBench v1.2 total time taken by different tools for common 66 kernels on logarithmic scale

Results: OmpSCR v2.0 race conditions

Table: Number of Races detected in OmpSCR v2.0 benchmark (CT is Compilation Timeout)

Kernel	LLOV	HELGRIND	DRD	TSAN	ARCHER	SWORD
Manually verified kernels with data races						
c_loopA.badSolution	1	1	1	1	1	1
c_loopA.solution2	1	1	1	1	1	0
c_loopA.solution3	1	1	1	1	1	0
c_loopB.badSolution1	1	1	1	1	1	1
c_loopB.badSolution2	1	1	1	1	1	1
c_loopB.pipelineSolution	1	1	1	1	1	0
c_md	1	2	2	2	1	CT
c_lu	1	1	1	1	1	0
Manually verified race free kernels						
c_loopA.solution1	0	2	1	2	1	0
c_mandel	0	1	0	1	1	0
c_pi	0	1	0	1	1	0
c_jacobi01	1	2	1	0	0	CT
c_jacobi02	1	1	1	0	0	CT
c_jacobi03	0	1	1	0	0	CT
Unverified kernels						
c_fft	1	1	1	1	1	CT
c_fft6	1	1	0	1	1	CT
c_qsort	0	1	1	1	1	CT
c_GraphSearch	0	0	0	0	0	0
cpp_qsomp1	0	0	0	0	0	0
cpp_qsomp2	0	0	0	0	0	0
cpp_qsomp3	0	0	0	0	0	0
cpp_qsomp4	0	0	0	0	0	0
cpp_qsomp5	0	0	0	0	0	0
cpp_qsomp6	0	0	0	0	0	0
cpp_qsomp7	0	0	0	0	0	0

Results: OmpSCR v2.0 runtime

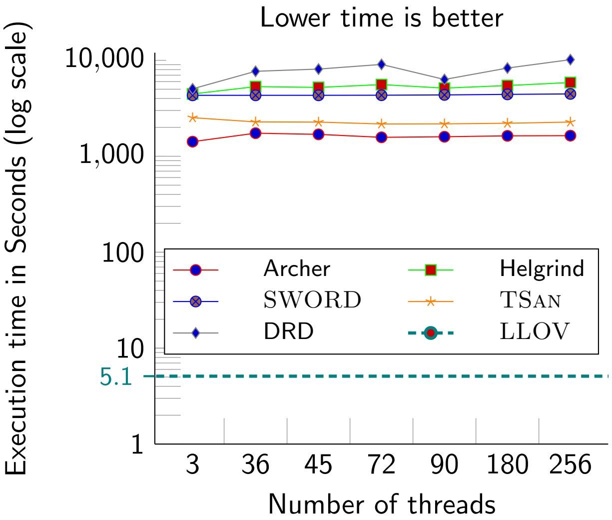

Figure: OmpSCR v2.0 total execution time by different tools on logarithmic scale

An implementation of DataRaceBench C/C++ v1.2 [LLSK18] in FORTRAN 95.

- Converted 92 (out of 116) C/C++ kernels to FORTRAN
- Demonstrate that LLOV is language agnostic
- Already open-sourced this benchmark [KSB19]

Results: DataRaceBench FORTRAN statistics

Table: Maximum number of Races reported by different tools in DataRaceBench FORTRAN

Tools	Race: Yes		Race: No		Coverage/92
	TP	FN	TN	FP	
HELGRIND	**46**	6	4	36	92
VALGRIND DRD	45	7	**21**	19	92
LLOV	34	6	19	5	64

Table of Contents

OpenMP v4.5 Pragma Handling Status: Various Tools

Table: Comparison of OpenMP pragma handling by OpenMP aware tools. (Y for Yes, N for No)

OpenMP Pragma	LLOV	PolyOMP	DRACO	SWORD
#pragma omp parallel	Y	Y	Y	Y
#pragma omp for	Y	Y	Y	Y
#pragma omp parallel for	Y	Y	Y	Y
#pragma omp atomic	Y	N	N	Y
#pragma omp threadprivate	Y	N	N	N
#pragma omp master	Y	N	N	Y
#pragma omp single	Y	N	N	Y
#pragma omp simd	Y	N	Y	N
#pragma omp parallel for simd	Y	N	Y	N
#pragma omp distribute	Y	N	N	N
#pragma omp ordered	Y	N	N	N
#pragma omp critical	Y	N	N	Y
#pragma omp parallel sections	N	N	N	Y
#pragma omp sections	N	N	N	Y
#pragma omp declare reduction	N	N	N	N
#pragma omp task	N	N	N	N
#pragma omp taskgroup	N	N	N	N
#pragma omp taskloop	N	N	N	N
#pragma omp taskwait	N	N	N	N
#pragma omp teams	N	N	N	N
#pragma omp barrier	N	N	N	Y
#pragma omp target map	N	N	N	N

Table of Contents

Possible Extensions to LLOV

Working on

- Use approximate dependece analysis (LAI) [Gro19] of LLVM

Possible Extensions to LLOV

Working on

- Use approximate dependece analysis (LAI) [Gro19] of LLVM
- Increase coverage- handle more OpenMP pragmas

Possible Extensions to LLOV

Working on

- Use approximate dependece analysis (LAI) [Gro19] of LLVM
- Increase coverage- handle more OpenMP pragmas
- Use May-Happen-in-Parallel analysis for race detection

Contributions Welcome!!

Open source links:

- DataRaceBench FORTRAN: https://github.com/IITH-Compilers/drb_fortran
- LLOV: Please drop me an email at cs14mtech11017@iith.ac.in

We welcome your contributions in any form.

Thanks and Acknowledgements

Johannes Doerfert
Tobias Grosser
GSoC mentors for "Polly as a pass in LLVM"
LLVM Community

References I

Simone Atzeni, Ganesh Gopalakrishnan, Zvonimir Rakamaric, Dong H Ahn, Ignacio Laguna, Martin Schulz, Gregory L Lee, Joachim Protze, and Matthias S Müller.
Archer: effectively spotting data races in large openmp applications.
In *Parallel and Distributed Processing Symposium, 2016 IEEE International*, pages 53–62, Chicago, IL, USA, 2016. IEEE, IEEE.

Simone Atzeni, Ganesh Gopalakrishnan, Zvonimir Rakamaric, Ignacio Laguna, Gregory L Lee, and Dong H Ahn.
Sword: A bounded memory-overhead detector of openmp data races in production runs.
In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 845–854, Vancouver, BC, Canada, 2018. IEEE, IEEE.

Vamshi Basupalli, Tomofumi Yuki, Sanjay Rajopadhye, Antoine Morvan, Steven Derrien, Patrice Quinton, and David Wonnacott.
ompverify: polyhedral analysis for the openmp programmer.
In *International Workshop on OpenMP*, pages 37–53, Berlin, Heidelberg, 2011. Springer, Springer Berlin Heidelberg.

P. Chatarasi, J. Shirako, and V. Sarkar.
Polyhedral optimizations of explicitly parallel programs.
In *2015 International Conference on Parallel Architecture and Compilation (PACT)*, pages 213–226, San Francisco, CA, USA, Oct 2015. IEEE.

References II

A.J. Dorta.
OpenMP Source Code Repository.
https://sourceforge.net/projects/ompscr/files/OmpSCR/OmpSCR%20Full%20Distribution%20v2.0/, 2004.
[Online; accessed 19-May-2019].

A. J. Dorta, C. Rodriguez, and F. de Sande.
The openmp source code repository.
In *13th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 244–250, Washington, DC, USA, Feb 2005. IEEE Computer Society.

Yizi Gu and John Mellor-Crummey.
Dynamic data race detection for openmp programs.
In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '18, pages 61:1–61:12, Piscataway, NJ, USA, 2018. IEEE Press.

LLVM Developer Group.
Loop Access Info, Class Reference .
https://llvm.org/doxygen/classllvm_1_1LoopAccessInfo.html, 2019.
[Online; accessed 08-May-2019].

References III

Pankaj Kukreja, Himanshu Shukla, and Utpal Bora.
DataRaceBench FORTRAN.
https://github.com/IITH-Compilers/drb_fortran, 2019.
[Online; accessed 19-October-2019].

Chunhua Liao, Pei-Hung Lin, Joshua Asplund, Markus Schordan, and Ian Karlin.
DataRaceBench v1.2.0.
https://github.com/LLNL/dataracebench, 2018.
[Online; accessed 19-May-2019].

Chunhua Liao, Pei-Hung Lin, Markus Schordan, and Ian Karlin.
A semantics-driven approach to improving dataracebench's openmp standard coverage.
In Bronis R. de Supinski, Pedro Valero-Lara, Xavier Martorell, Sergi Mateo Bellido, and Jesus Labarta, editors, *Evolving OpenMP for Evolving Architectures*, pages 189–202, Cham, 2018. Springer International Publishing.

Konstantin Serebryany and Timur Iskhodzhanov.
Threadsanitizer: Data race detection in practice.
In *Proceedings of the Workshop on Binary Instrumentation and Applications*, WBIA '09, pages 62–71, New York, NY, USA, 2009. ACM.

References IV

Valgrind-project.
DRD: a thread error detector.
http://valgrind.org/docs/manual/drd-manual.html, 2007.
[Online; accessed 08-May-2019].

Valgrind-project.
Helgrind: a thread error detector.
http://valgrind.org/docs/manual/hg-manual.html, 2007.
[Online; accessed 08-May-2019].

Fangke Ye, Markus Schordan, Chunhua Liao, Pei-Hung Lin, Ian Karlin, and Vivek Sarkar.
Using polyhedral analysis to verify openmp applications are data race free.
In *2018 IEEE/ACM 2nd International Workshop on Software Correctness for HPC Applications (Correctness)*, pages 42–50, Dallas, TX, USA, 2018. IEEE, IEEE.

Thank You!

LLOV: Race Detection Algorithm

Algorithm 1: Race Detection Algorithm

Input: *L*

Output: result

```
Function isRaceFree(L):
    SCoP = ConstructSCoP(L) ;
    RDG = ComputeDependences(SCoP) ;
    depth = GetLoopDepth(L) ;
    if isParallel(RDG, depth) then
        result = "Program is race free." ;
    else
        result = "Data Race detected." ;
    return result
End Function
```

Algorithm 2: Algorithm to check parallelism

Input: *RDG*, *dim*

Output: True/False

```
Function isParallel(RDG, dim):
    if RDG is Empty then
        return True ;
    else
        Flag = True;
        while Dependence D in RDG do
            D' = Project Out first dim dimensions from D ;
            if D' is Empty then
                continue ;
            else
                Flag = False ;
                break ;
        return Flag ;
End Function
```

- **True Positive (TP):** If the evaluation tool correctly detects a data race present in the kernel it is a True Positive test result. A higher number of true positives represents a better tool.
- **True Negative (TN):** If the benchmark does not contain a race and the tool declares it as race-free, then it is a true negative case. A higher number of true negatives represents a better tool.
- **False Positives (FP):** If the benchmark does not contain any race, but the tool reports a race condition, it is a false positive. False Positives should be as low as possible.
- **False Negatives (FN):** False Negative test result is obtained when the tool fails to detect a known race in the benchmark. These are the cases that are missed by the tool. A lower number of false negatives are desirable.

Terminology II

- **Precision :** Precision is the measure of closeness of the outcomes of prediction. Thus, a higher value of precision represents that the tool will more often than not identify a race condition when it exists.
$Precision = \frac{TP}{TP + FP}$
- **Recall :** Recall gives the total number of cases detected out of the maximum data races present. A higher recall value means that there are less chances that a data race is missed by the tool. It is also called true positive rate (TPR).
$Recall = \frac{TP}{TP + FN}$
- **Accuracy :** Accuracy gives the chances of correct reports out of all the reports, as the name suggests. A higher value of accuracy is always desired and gives overall measure of the efficacy of the tool.
$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$

Terminology III

- **F1 Score :** The harmonic mean of precision and recall is called the F1 score. An F1 score of 1 can be achieved in the best case when both precision and recall are perfect. The worst case F1 score is 0 when either precision or recall is 0.
  $F1\ Score = 2 * \frac{Precision\ *\ Recall}{Precision\ +\ Recall}$
- **Diagnostic odds ratio (DOR) :** It is the ratio of the positive likelihood ratio (LR+) to the negative likelihood ratio (LR−).
  $DOR = \frac{LR+}{LR-}$ where,
  Positive Likelihood Ratio $(LR+) = \frac{TPR}{FPR}$,
  Negative Likelihood Ratio $(LR-) = \frac{FNR}{TNR}$,
  True Positive Rate $(TPR) = \frac{TP}{TP\ +\ FN}$,
  False Positive Rate $(FPR) = \frac{FP}{FP\ +\ TN}$,
  False Negative Rate $(FNR) = \frac{FN}{FN\ +\ TP}$ and
  True Negative Rare $(TNR) = \frac{TN}{TN\ +\ FP}$

Terminology IV

DOR is the measure of the ratio of the odds of race detection being positive given that the test case has a data race, to the odds of race detection being positive given the test case does not have a race.