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

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Utpal Bora* Santanu Das Pankaj Kukreja Saurabh Joshi Ramakrishna Upadrasta Sanjay Rajopadhye

*PhD Student
Computer Science and Engineering
IIT Hyderabad, India

February 11, 2021

Common data 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;
    }
```

Listing 1: DRB021: OpenMP Worksharing construct with data race

Common data 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];
   }
}
```

Listing 2: DRB038: Example with Loop Carried Dependence

Common data race conditions in OpenMP programs

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

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

Listing 3: DRB024: Example with SIMD data race

Common data 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;
  }
```

Listing 4: DRB013: Example with data race due to improper synchronization

Common data 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;
  }
```

Listing 5: Control flow dependent on number of threads

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)
- 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)
- can handle FORTRAN as well as C/C++
- 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)
- can handle FORTRAN as well as C/C++
- uses Polyhedral framework, Polly, of LLVM
- can conservatively state when a program is data 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)
- can handle FORTRAN as well as C/C++
- uses Polyhedral framework, Polly, of LLVM
- can conservatively state when a program is data race free
- is capable of generating task graphs of OpenMP constructs

LLOV: Flow Diagram

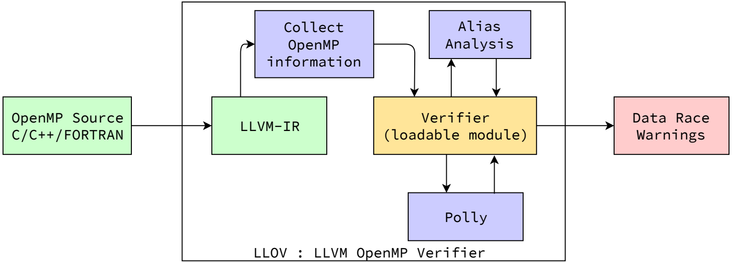

Figure 1: Flow Diagram of LLVM OpenMP Verifier (LLOV)

LLOV: In-Memory Representation

```
#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;
  }
```

LLOV: In-Memory Representation

```c
#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;
  }
```

```
Directive:  OMP_Parallel
  Variables:
    Private:   %.omp.ub = alloca i32, align 4
    Private:   %.omp.lb = alloca i32, align 4
    Shared: i32* %i
    Shared: i32* %len
    Firstprivate: i64 %vla
    Shared: i32* %a
    Shared: i32* %b
    Shared: i32* %error
    Private:   %.omp.stride = alloca i32, align 4
    Private:   %.omp.is_last = alloca i32, align 4
  Child Directives:
  1:  Directive:    OMP_Workshare_Loop
    Schedule type : Static Schedule (auto-chunked)
  2:  Directive:    OMP_Workshare_single
  3:  Directive:    OMP_Barrier
```

Listing 7: In-memory representation of a directive

LLOV: In-Memory Representation

```
#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;
  }
```

```
Directive:  OMP_Parallel
  Variables:
    Private:   %.omp.ub = alloca i32, align 4
    Private:   %.omp.lb = alloca i32, align 4
    Shared: i32* %i
    Shared: i32* %len
    Firstprivate: i64 %vla
    Shared: i32* %a
    Shared: i32* %b
    Shared: i32* %error
    Private:   %.omp.stride = alloca i32, align 4
    Private:   %.omp.is_last = alloca i32, align 4
  Child Directives:
  1:  Directive:    OMP_Workshare_Loop
    Schedule type : Static Schedule (auto-chunked)
  2:  Directive:    OMP_Workshare_single
  3:  Directive:    OMP_Barrier
```

```
<Directive> ::= <Dtype> [ Sched ] { <Var> } { <Directive> }
<Dtype>     ::= parallel | for | simd
                | workshare | single
                | master | critical
<Var>       ::= <Vtype> val
<Vtype>     ::= private | firstprivate
                | shared | lastprivate
                | reduction | threadprivate
<Sched>     ::= [ <modifier> ] [ ordered ] <Stype> <chunk>
<modifier>  ::= monotonic | nonmonotonic
<Stype>     ::= static | dynamic | guided | auto | runtime
<chunk>     ::= positive-int-const
```

Listing 7: In-memory representation of a directive

LLOV: Polyhedral Representation

```
for (i=0; i<m ; i++) {
#pragma omp parallel for
   for (j=1; j<n ;j++) {
S0:    b[i][j] = b[i][j-1];
   }
 }
```

Iteration Domain : $\mathbf{I} = \{\mathtt{S0}(i,j) : 0 \leq i \leq m-1 \wedge 1 \leq j \leq n-1\}$

Schedule : $\mathbf{S} = \{\mathtt{S0}(i,j) \to (i,j)\} \cap_{dom} \mathbf{I}$

Access Map : $\mathbf{A} = \{\mathtt{S0}(i,j) \to \mathtt{M}(i,j); \mathtt{S0}(i,j) \to \mathtt{M}(i,j-1)\}$

Dependences : $\mathbf{D} = \{\mathtt{S0}(i,j) \to (i,j-1) : 0 \leq i \leq m-1 \wedge 1 \leq j \leq n-1\}$

LLOV: Polyhedral Representation

```
for (i=0; i<m ; i++) {
#pragma omp parallel for
   for (j=1; j<n ;j++) {
S0:    b[i][j] = b[i][j-1];
   }
 }
```

Iteration Domain : $\mathbf{I} = \{\texttt{S0}(i,j) : \boxed{0 \leq i} \leq m-1 \land 1 \leq j \leq n-1\}$

Schedule : $\mathbf{S} = \{\texttt{S0}(i,j) \to (i,j)\} \cap_{dom} \mathbf{I}$

Access Map : $\mathbf{A} = \{\texttt{S0}(i,j) \to \texttt{M}(i,j); \texttt{S0}(i,j) \to \texttt{M}(i,j-1)\}$

Dependences : $\mathbf{D} = \{\texttt{S0}(i,j) \to (i,j-1) : 0 \leq i \leq m-1 \land 1 \leq j \leq n-1\}$

LLOV: Polyhedral Representation

```c
for (i=0; i<m ; i++) {
#pragma omp parallel for
   for (j=1; j<n ;j++) {
S0:    b[i][j] = b[i][j-1];
   }
 }
```

Iteration Domain : $\mathbf{I} = \{\texttt{S0}(i,j) : 0 \leq \boxed{i \leq m-1} \wedge 1 \leq j \leq n-1\}$

Schedule : $\mathbf{S} = \{\texttt{S0}(i,j) \to (i,j)\} \cap_{dom} \mathbf{I}$

Access Map : $\mathbf{A} = \{\texttt{S0}(i,j) \to \texttt{M}(i,j); \texttt{S0}(i,j) \to \texttt{M}(i,j-1)\}$

Dependences : $\mathbf{D} = \{\texttt{S0}(i,j) \to (i,j-1) : 0 \leq i \leq m-1 \wedge 1 \leq j \leq n-1\}$

LLOV: Polyhedral Representation

```
for (i=0; i<m ; i++) {
#pragma omp parallel for
   for (j=1; j<n ;j++) {
S0:    b[i][j] = b[i][j-1];
   }
 }
```

Iteration Domain : $\mathbf{I} = \{\texttt{S0}(i,j) : 0 \leq i \leq m-1 \wedge \boxed{1 \leq j} \leq n-1\}$

Schedule : $\mathbf{S} = \{\texttt{S0}(i,j) \rightarrow (i,j)\} \cap_{dom} \mathbf{I}$

Access Map : $\mathbf{A} = \{\texttt{S0}(i,j) \rightarrow \texttt{M}(i,j); \texttt{S0}(i,j) \rightarrow \texttt{M}(i,j-1)\}$

Dependences : $\mathbf{D} = \{\texttt{S0}(i,j) \rightarrow (i,j-1) : 0 \leq i \leq m-1 \wedge 1 \leq j \leq n-1\}$

LLOV: Polyhedral Representation

```
for (i=0; i<m ; i++) {
#pragma omp parallel for
   for (j=1; j<n ;j++) {
S0:    b[i][j] = b[i][j-1];
   }
 }
```

Iteration Domain : $\mathbf{I} = \{\mathtt{S0}(i,j) : 0 \leq i \leq m-1 \wedge 1 \leq \boxed{j \leq n-1}\}$

Schedule : $\mathbf{S} = \{\mathtt{S0}(i,j) \rightarrow (i,j)\} \cap_{dom} \mathbf{I}$

Access Map : $\mathbf{A} = \{\mathtt{S0}(i,j) \rightarrow \mathtt{M}(i,j); \mathtt{S0}(i,j) \rightarrow \mathtt{M}(i,j-1)\}$

Dependences : $\mathbf{D} = \{\mathtt{S0}(i,j) \rightarrow (i,j-1) : 0 \leq i \leq m-1 \wedge 1 \leq j \leq n-1\}$

LLOV: Polyhedral Representation

```
for (i=0; i<m ; i++) {
#pragma omp parallel for
   for (j=1; j<n ;j++) {
S0:    b[i][j] = b[i][j-1];
   }
 }
```

Iteration Domain : $\mathbf{I} = \{\mathtt{S0}(i,j) : 0 \le i \le m-1 \wedge 1 \le j \le n-1\}$

Schedule : $\mathbf{S} = \{\mathtt{S0}(i,j) \to (i,j)\} \cap_{dom} \mathbf{I}$

Access Map : $\mathbf{A} = \{\mathtt{S0}(i,j) \to \boxed{\mathtt{M}(i,j)}; \mathtt{S0}(i,j) \to \mathtt{M}(i,j-1)\}$

Dependences : $\mathbf{D} = \{\mathtt{S0}(i,j) \to (i,j-1) : 0 \le i \le m-1 \wedge 1 \le j \le n-1\}$

LLOV: Polyhedral Representation

```
for (i=0; i<m ; i++) {
#pragma omp parallel for
   for (j=1; j<n ;j++) {
S0:    b[i][j] = b[i][j-1];
   }
 }
```

Iteration Domain : $\mathbf{I} = \{S0(i,j) : 0 \leq i \leq m-1 \wedge 1 \leq j \leq n-1\}$

Schedule : $\mathbf{S} = \{S0(i,j) \to (i,j)\} \cap_{dom} \mathbf{I}$

Access Map : $\mathbf{A} = \{S0(i,j) \to \mathtt{M}(i,j); S0(i,j) \to \boxed{\mathtt{M}(i,j-1)}\}$

Dependences : $\mathbf{D} = \{S0(i,j) \to (i,j-1) : 0 \leq i \leq m-1 \wedge 1 \leq j \leq n-1\}$

```
for (i=0; i<m ; i++) {
#pragma omp parallel for
   for (j=1; j<n ;j++) {
S0:    b[i][j] = b[i][j-1];
   }
 }
```

Iteration Domain : $\mathbf{I} = \{\mathtt{S0}(i,j) : 0 \leq i \leq m-1 \wedge 1 \leq j \leq n-1\}$

Schedule : $\mathbf{S} = \{\mathtt{S0}(i,j) \to (i,j)\} \cap_{dom} \mathbf{I}$

Access Map : $\mathbf{A} = \{\mathtt{S0}(i,j) \to \mathtt{M}(i,j); \mathtt{S0}(i,j) \to \mathtt{M}(i,j-1)\}$

Dependences : $\mathbf{D} = \{\boxed{\mathtt{S0}(i,j) \to (i,j-1)} : 0 \leq i \leq m-1 \wedge 1 \leq j \leq n-1\}$

LLOV: Polyhedral Representation

```c
for (i=0; i<m ; i++) {
#pragma omp parallel for
   for (j=1; j<n ;j++) {
S0:    b[i][j] = b[i][j-1];
   }
 }
```

Iteration Domain : $\mathbf{I} = \{\texttt{S0}(i,j) : \boxed{0 \le i \le m-1 \wedge 1 \le j \le n-1}\}$

Schedule : $\mathbf{S} = \{\texttt{S0}(i,j) \to (i,j)\} \cap_{dom} \mathbf{I}$

Access Map : $\mathbf{A} = \{\texttt{S0}(i,j) \to \texttt{M}(i,j); \texttt{S0}(i,j) \to \texttt{M}(i,j-1)\}$

Dependences : $\mathbf{D} = \{\texttt{S0}(i,j) \to (i,j-1) : \boxed{0 \le i \le m-1 \wedge 1 \le j \le n-1}\}$

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];
   }
}
```

Listing 10: Example with Loop Carried Dependence

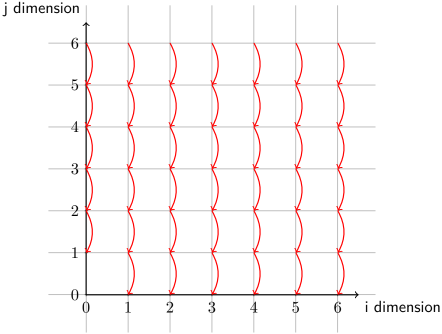

Figure 2: Dependence Polyhedra

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];
   }
}
```

Listing 11: Example with Loop Carried Dependence

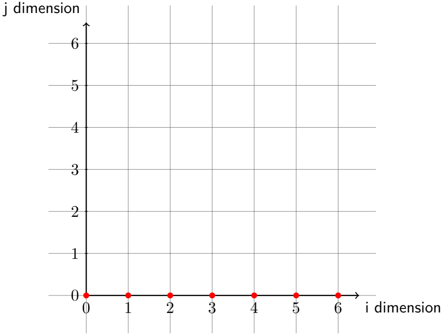

Figure 3: Projection of the Dependence Polyhedra on i-dimension

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

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

Listing 12: Example with Loop Carried Dependence

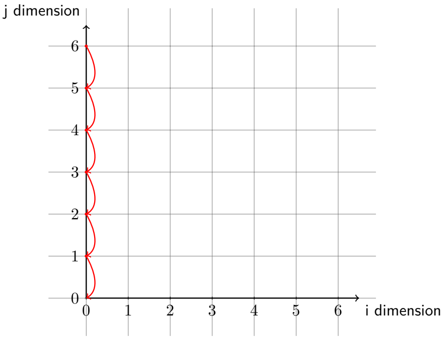

Figure 4: Projection of the Dependence Polyhedra on j-dimension

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

Results: Experimental Setup

Benchmarks:

- DataRaceBench C/C++ v1.2 [Liao et al., 2018a, Liao et al., 2018b]
- OmpSCR v2.0 [Dorta et al., 2004, Dorta et al., 2005]
- DataRaceBench FORTRAN [Kukreja et al., 2019]

Results: Experimental Setup

Benchmarks:

- DataRaceBench C/C++ v1.2 [Liao et al., 2018a, Liao et al., 2018b]
- OmpSCR v2.0 [Dorta et al., 2004, Dorta et al., 2005]
- DataRaceBench FORTRAN [Kukreja et al., 2019]

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

Tools	Source	Version / Commit
HELGRIND [Valgrind-project, 2007b]	Valgrind	3.13.0
VALGRIND DRD [Valgrind-project, 2007a]	Valgrind	3.13.0
TSAN-LLVM [Serebryany and Iskhodzhanov, 2009]	LLVM	6.0.1
ARCHER [Atzeni et al., 2016]	git master branch	fc17353
SWORD [Atzeni et al., 2018]	git master branch	7a08f3c

Results: DataRaceBench v1.2 comparison

Table 2: 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	48	2	**36**	5	91

Results: DataRaceBench v1.2 comparison

Table 2: 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	48	2	**36**	5	91

Table 3: Maximum number of Races reported by different tools in common 61 kernels of DataRaceBench 1.2

Tools	Race: Yes		Race: No		Coverage/61
	TP	FN	TN	FP	
Helgrind	**42**	1	2	16	61
Valgrind DRD	**42**	1	12	6	61
TSan-LLVM	**42**	1	2	16	61
Archer	**42**	1	2	16	61
SWORD	**42**	1	**17**	1	61
LLOV	**42**	1	16	2	61

Results: DataRaceBench v1.2 statistics

Table 4: Performance 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	**0.91**	**0.96**	**0.92**	**0.93**	**172.80**

Results: DataRaceBench v1.2 statistics

Table 4: Performance 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	**0.91**	**0.96**	**0.92**	**0.93**	**172.80**

Table 5: Performance of the tools on common 61 kernels of DataRaceBench 1.2

Tools	Precision	Recall	Accuracy	F1 Score	Diagnostic odds ratio
HELGRIND	0.72	**0.98**	0.72	0.83	5.25
VALGRIND DRD	0.88	**0.98**	0.89	0.92	84.00
TSAN-LLVM	0.72	**0.98**	0.72	0.83	5.25
ARCHER	0.72	**0.98**	0.72	0.83	5.25
SWORD	**0.98**	**0.98**	**0.97**	**0.98**	**714.00**
LLOV	**0.95**	**0.98**	**0.95**	**0.97**	**336.00**

Results: DataRaceBench v1.2 runtime

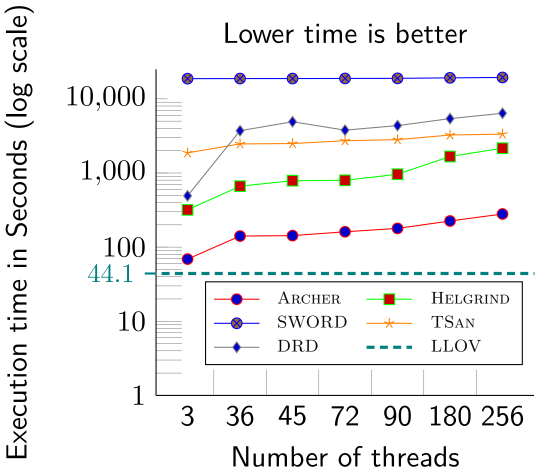

Figure 5: DataRaceBench v1.2 total execution time by different tools on logarithmic scale

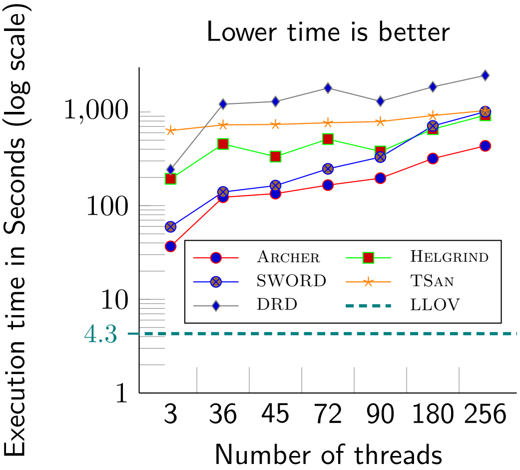

Figure 6: DataRaceBench v1.2 total time taken by different tools for common 61 kernels on logarithmic scale

Results: OmpSCR v2.0 comparison

Table 6: Comparison of different tools on OmpSCR v2.0

Tools	Race: Yes		Race: No		Coverage/14
	TP	FN	TN	FP	
HELGRIND	**8**	0	0	9	14
VALGRIND DRD	**8**	0	2	5	14
TSAN-LLVM	7	1	2	6	14
ARCHER	7	1	2	4	14
SWORD	3	4	**3**	0	10
LLOV	4	1	2	5	10

Results: OmpSCR v2.0 runtime

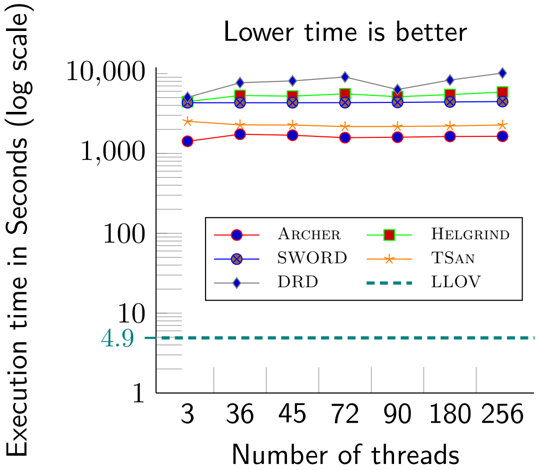

Figure 7: OmpSCR v2.0 total execution time by different tools on logarithmic scale

DataRaceBench FORTRAN

An implementation of DataRaceBench C/C++ v1.2 [Liao et al., 2018b] in FORTRAN 95.

- Converted 92 (out of 116) C/C++ kernels to FORTRAN
- Demonstrate that LLOV is language agnostic
- Already open-sourced this benchmark [Kukreja et al., 2019]

Results: DataRaceBench FORTRAN statistics

Table 7: 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	36	7	19	5	67

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

LLOV is freely available for download.
Link: https://github.com/utpalbora/llov
Blog: https://compilers.cse.iith.ac.in/projects/llov/

utpalbora.com

Open source links:

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

Contributions Welcome!!

We welcome your contributions in any form. **Thank You!**

References I

Atzeni, S., Gopalakrishnan, G., Rakamaric, Z., Ahn, D. H., Laguna, I., Schulz, M., Lee, G. L., Protze, J., and Müller, M. S. (2016).
Archer: effectively spotting data races in large openmp applications.
In *Parallel and Distributed Processing Symposium, 2016 IEEE International*, pages 53–62, Chicago, IL, USA. IEEE, IEEE.

Atzeni, S., Gopalakrishnan, G., Rakamaric, Z., Laguna, I., Lee, G. L., and Ahn, D. H. (2018).
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. IEEE, IEEE.

Dorta, A. J., Rodriguez, C., and de Sande, F. (2004).
OpenMP Source Code Repository.
`https://sourceforge.net/projects/ompscr/files/OmpSCR/`.
[Online; accessed 19-May-2019].

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

References II

Kukreja, P., Shukla, H., and Bora, U. (2019).
DataRaceBench FORTRAN.
https://github.com/IITH-Compilers/drb_fortran.
[Online; accessed 19-October-2019].

Liao, C., Lin, P.-H., Asplund, J., Schordan, M., and Karlin, I. (2018a).
DataRaceBench v1.2.0.
https://github.com/LLNL/dataracebench.
[Online; accessed 19-May-2019].

Liao, C., Lin, P.-H., Schordan, M., and Karlin, I. (2018b).
A semantics-driven approach to improving dataracebench's openmp standard coverage.
In *Evolving OpenMP for Evolving Architectures*, pages 189–202, Cham. Springer International Publishing.

Serebryany, K. and Iskhodzhanov, T. (2009).
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. ACM.

References III

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

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