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

1 Motivation for LLOV

2 Architecture and Methodology

3 Results

4 Current Status

5 Extensions
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

```c
#include <omp.h>

int len = 10;

# 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

```c
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

```c
#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

<table>
<thead>
<tr>
<th>Tools</th>
<th>Infrastructure</th>
<th>Analysis Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Helgrind</strong> [Vp07b]</td>
<td>Valgrind</td>
<td>Dynamic</td>
</tr>
<tr>
<td><strong>Valgrind DRD</strong> [Vp07a]</td>
<td>Valgrind</td>
<td>Dynamic</td>
</tr>
<tr>
<td><strong>TSan</strong> [SI09]</td>
<td>LLVM/GCC</td>
<td>Dynamic</td>
</tr>
<tr>
<td><strong>Archer</strong> [AGR\textsuperscript+16]</td>
<td>LLVM</td>
<td>Hybrid</td>
</tr>
<tr>
<td><strong>Sword</strong> [AGR\textsuperscript+18]</td>
<td>LLVM</td>
<td>Dynamic</td>
</tr>
<tr>
<td><strong>Romp</strong> [GMC18]</td>
<td>Dyninst</td>
<td>Dynamic</td>
</tr>
<tr>
<td><strong>Polyomp</strong> [CSS15]</td>
<td>ROSE</td>
<td>Static</td>
</tr>
<tr>
<td><strong>Draco</strong> [YSL\textsuperscript+18]</td>
<td>ROSE</td>
<td>Static</td>
</tr>
<tr>
<td><strong>OmpVerify</strong> [BYR\textsuperscript+11]</td>
<td>AlphaZ</td>
<td>Static</td>
</tr>
</tbody>
</table>

There is still need for a static OpenMP data race checker in LLVM.
**Table: OpenMP Race Detection Tools: A Short Survey**

<table>
<thead>
<tr>
<th>Tools</th>
<th>Infrastructure</th>
<th>Analysis Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEGrind [Vp07b]</td>
<td>Valgrind</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Valgrind DRD [Vp07a]</td>
<td>Valgrind</td>
<td>Dynamic</td>
</tr>
<tr>
<td>TSan [SI09]</td>
<td>LLVM/GCC</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Archer [AGR+16]</td>
<td>LLVM</td>
<td>Hybrid</td>
</tr>
<tr>
<td>Sword [AGR+18]</td>
<td>LLVM</td>
<td>Dynamic</td>
</tr>
<tr>
<td>ROMP [GMC18]</td>
<td>Dyninst</td>
<td>Dynamic</td>
</tr>
<tr>
<td>PolyOMP [CSS15]</td>
<td>ROSE</td>
<td>Static</td>
</tr>
<tr>
<td>Draco [YSL+18]</td>
<td>ROSE</td>
<td>Static</td>
</tr>
<tr>
<td>OmpVerify [BYR+11]</td>
<td>AlphaZ</td>
<td>Static</td>
</tr>
</tbody>
</table>

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

LLVM is an attempt to bridge this gap and move towards a fast, language agnostic, robust, 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
- 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

1 Motivation for LLOV

2 Architecture and Methodology

3 Results

4 Current Status

5 Extensions
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 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)

```c
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)

```c
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**.
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

<table>
<thead>
<tr>
<th>Tools</th>
<th>Source</th>
<th>Version / Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Helgrind [Vp07b]</td>
<td>Valgrind</td>
<td>3.13.0</td>
</tr>
<tr>
<td>Valgrind DRD [Vp07a]</td>
<td>Valgrind</td>
<td>3.13.0</td>
</tr>
<tr>
<td>TSan-LLVM [SI09]</td>
<td>LLVM</td>
<td>6.0.1</td>
</tr>
<tr>
<td>Archer [AGR+16]</td>
<td>git master branch</td>
<td>fc17353</td>
</tr>
<tr>
<td>SWORD [AGR+18]</td>
<td>git master branch</td>
<td>7a08f3c</td>
</tr>
<tr>
<td>ROMP [GMC18]</td>
<td>git master branch</td>
<td>6a0ad6d</td>
</tr>
</tbody>
</table>
### Results: DataRaceBench v1.2 comparison

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

<table>
<thead>
<tr>
<th>Tools</th>
<th>Race: Yes</th>
<th>Race: No</th>
<th>Coverage/116</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TP</td>
<td>FN</td>
<td>TN</td>
</tr>
<tr>
<td><strong>Helgrind</strong></td>
<td>56</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td><strong>Valgrind DRD</strong></td>
<td>56</td>
<td>3</td>
<td>26</td>
</tr>
<tr>
<td><strong>TSan-LLVM</strong></td>
<td>57</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>Archer</strong></td>
<td>56</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td><strong>SWORD</strong></td>
<td>47</td>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td><strong>LLOV</strong></td>
<td>45</td>
<td>3</td>
<td><strong>28</strong></td>
</tr>
</tbody>
</table>
### Table: Maximum number of Races reported by different tools in DataRaceBench 1.2

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

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

<table>
<thead>
<tr>
<th>Tools</th>
<th>Race: Yes</th>
<th>Race: No</th>
<th>Coverage/116</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TP  FN</td>
<td>TN  FP</td>
<td></td>
</tr>
<tr>
<td>Helgrind</td>
<td>46  1</td>
<td>2  17</td>
<td>66</td>
</tr>
<tr>
<td>Valgrind DRD</td>
<td>46  1</td>
<td>13  6</td>
<td>66</td>
</tr>
<tr>
<td>TSan-LLVM</td>
<td>46  1</td>
<td>2  17</td>
<td>66</td>
</tr>
<tr>
<td>Archer</td>
<td>46  1</td>
<td>2  17</td>
<td>66</td>
</tr>
<tr>
<td>SWORD</td>
<td>46  1</td>
<td>18  1</td>
<td>66</td>
</tr>
<tr>
<td>LLOV</td>
<td>44  3</td>
<td>16  3</td>
<td>66</td>
</tr>
</tbody>
</table>
### Table: Precision, Recall and Accuracy of the tools on DataRaceBench 1.2

<table>
<thead>
<tr>
<th>Tools</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
<th>F1 Score</th>
<th>Diagnostic odds ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Helgrind</td>
<td>0.50</td>
<td>0.95</td>
<td>0.50</td>
<td>0.66</td>
<td>0.68</td>
</tr>
<tr>
<td>Valgrind DRD</td>
<td>0.64</td>
<td>0.95</td>
<td>0.71</td>
<td>0.77</td>
<td>15.66</td>
</tr>
<tr>
<td>TSan-LLVM</td>
<td>0.51</td>
<td>0.97</td>
<td>0.51</td>
<td>0.67</td>
<td>1.04</td>
</tr>
<tr>
<td>Archer</td>
<td>0.50</td>
<td>0.95</td>
<td>0.50</td>
<td>0.66</td>
<td>0.68</td>
</tr>
<tr>
<td>SWORD</td>
<td>0.92</td>
<td>0.92</td>
<td>0.90</td>
<td>0.92</td>
<td>70.50</td>
</tr>
<tr>
<td>LLOV</td>
<td>0.83</td>
<td>0.94</td>
<td>0.86</td>
<td>0.88</td>
<td>46.67</td>
</tr>
</tbody>
</table>
## Results: DataRaceBench v1.2 statistics

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

<table>
<thead>
<tr>
<th>Tools</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
<th>F1 Score</th>
<th>Diagnostic odds ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Helgrind</td>
<td>0.50</td>
<td>0.95</td>
<td>0.50</td>
<td>0.66</td>
<td>0.68</td>
</tr>
<tr>
<td>Valgrind DRD</td>
<td>0.64</td>
<td>0.95</td>
<td>0.71</td>
<td>0.77</td>
<td>15.66</td>
</tr>
<tr>
<td>TSan-LLVM</td>
<td>0.51</td>
<td>0.97</td>
<td>0.51</td>
<td>0.67</td>
<td>1.04</td>
</tr>
<tr>
<td>Archer</td>
<td>0.50</td>
<td>0.95</td>
<td>0.50</td>
<td>0.66</td>
<td>0.68</td>
</tr>
<tr>
<td>SWORD</td>
<td>0.92</td>
<td>0.92</td>
<td>0.90</td>
<td>0.92</td>
<td>70.50</td>
</tr>
<tr>
<td>LLOV</td>
<td>0.83</td>
<td>0.94</td>
<td>0.86</td>
<td>0.88</td>
<td>46.67</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Tools</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
<th>F1 Score</th>
<th>Diagnostic odds ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Helgrind</td>
<td>0.73</td>
<td>0.98</td>
<td>0.73</td>
<td>0.84</td>
<td>5.41</td>
</tr>
<tr>
<td>Valgrind DRD</td>
<td>0.88</td>
<td>0.98</td>
<td>0.89</td>
<td>0.93</td>
<td>99.67</td>
</tr>
<tr>
<td>TSan-LLVM</td>
<td>0.73</td>
<td>0.98</td>
<td>0.73</td>
<td>0.84</td>
<td>5.41</td>
</tr>
<tr>
<td>Archer</td>
<td>0.73</td>
<td>0.98</td>
<td>0.73</td>
<td>0.84</td>
<td>5.41</td>
</tr>
<tr>
<td>SWORD</td>
<td>0.98</td>
<td>0.98</td>
<td>0.97</td>
<td>0.98</td>
<td>828.00</td>
</tr>
<tr>
<td>LLOV</td>
<td>0.94</td>
<td>0.94</td>
<td>0.91</td>
<td>0.94</td>
<td>78.22</td>
</tr>
</tbody>
</table>
Results: DataRaceBench v1.2 runtime

![Graph showing execution time in seconds for different tools across varying numbers of threads.](image)

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

![Graph showing execution time in seconds for different tools across varying number of threads. Lower time is better.]

**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)

<table>
<thead>
<tr>
<th>Kernel</th>
<th>LLOV</th>
<th>Helgrind</th>
<th>DRD</th>
<th>TSan</th>
<th>Archer</th>
<th>SWORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manually verified kernels with data races</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c_loopA.badSolution</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>c_loopA.solution2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c_loopA.solution3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c_loopB.badSolution1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>c_loopB.badSolution2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>c_loopB.pipelineSolution</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c_md</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>CT</td>
</tr>
<tr>
<td>c_lu</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Manually verified race free kernels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c_loopA.solution1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c_mandel</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c_pi</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c_jacobi01</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>CT</td>
</tr>
<tr>
<td>c_jacobi02</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>CT</td>
</tr>
<tr>
<td>c_jacobi03</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>CT</td>
</tr>
<tr>
<td>Unverified kernels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c_fft</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>CT</td>
</tr>
<tr>
<td>c_fft6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>CT</td>
</tr>
<tr>
<td>c_qsort</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>CT</td>
</tr>
<tr>
<td>c_GraphSearch</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cpp_qsort1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cpp_qsort2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cpp_qsort3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cpp_qsort4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cpp_qsort5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cpp_qsort6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cpp_qsort7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Tools</th>
<th>Race: Yes</th>
<th>Race: No</th>
<th>Coverage/92</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TP</td>
<td>FN</td>
<td>TN</td>
</tr>
<tr>
<td>Helgrind</td>
<td>46</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Valgrind DRD</td>
<td>45</td>
<td>7</td>
<td><strong>21</strong></td>
</tr>
<tr>
<td>LLOV</td>
<td>34</td>
<td>6</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>Motivation for LLOV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Architecture and Methodology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Results</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Current Status</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Extensions</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**OpenMP v4.5 Pragma Handling Status: Various Tools**

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

<table>
<thead>
<tr>
<th>OpenMP Pragma</th>
<th>LLOV</th>
<th>POLYOMP</th>
<th>DRACO</th>
<th>SWORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp parallel</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>#pragma omp for</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>#pragma omp parallel for</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>#pragma omp atomic</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>#pragma omp threadprivate</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>#pragma omp master</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>#pragma omp single</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>#pragma omp simd</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>#pragma omp parallel for simd</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>#pragma omp distribute</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>#pragma omp ordered</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>#pragma omp critical</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>#pragma omp parallel sections</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>#pragma omp sections</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>#pragma omp declare reduction</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>#pragma omp task</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>#pragma omp taskgroup</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>#pragma omp taskloop</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>#pragma omp taskwait</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>#pragma omp teams</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>#pragma omp barrier</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>#pragma omp target map</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
Table of Contents

1. Motivation for LLOV
2. Architecture and Methodology
3. Results
4. Current Status
5. Extensions
Possible Extensions to LLOV

Working on

- Use approximate dependence analysis (LAI) [Gro19] of LLVM
Possible Extensions to LLOV

- Use approximate dependence analysis (LAI) [Gro19] of LLVM
- Increase coverage—handle more OpenMP pragmas
Possible Extensions to LLOV

- Use approximate dependence 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


A.J. Dorta.
OpenMP Source Code Repository.
[Online; accessed 19-May-2019].

A. J. Dorta, C. Rodriguez, and F. de Sande.
The openmp source code repository.

Yizi Gu and John Mellor-Crummey.
Dynamic data race detection for openmp programs.

LLVM Developer Group.
Loop Access Info, Class Reference.
[Online; accessed 08-May-2019].


Valgrind-project.  
DRD: a thread error detector.  
[Online; accessed 08-May-2019].

Valgrind-project.  
Helgrind: a thread error detector.  
[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(\( L \)):

1. \( SCoP = \) ConstructSCoP(\( L \))
2. \( RDG = \) ComputeDependences(\( SCoP \))
3. \( depth = \) GetLoopDepth(\( L \))
4. if isParallel(\( RDG, depth \)) then
   5. result = "Program is race free."
   6. return result
else
   7. result = "Data Race detected."
   8. return result
End Function

Algorithm 2: Algorithm to check parallelism

Input: \( RDG, \ dim \)
Output: True/False

Function isParallel(\( RDG, \ dim \)):

1. if \( RDG \) is Empty then
   2. return True
else
   3. Flag = True;
   4. while Dependence \( D \) in \( RDG \)
      do
         5. \( D' = \) Project Out first \( \dim \) dimensions from \( D \)
         6. if \( D' \) is Empty then
            7. continue
         else
            8. Flag = False
            9. break
   10. return Flag
End Function
Terminology I

- **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.
  \[ \text{Precision} = \frac{TP}{TP + FP} \]

- **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).
  \[ \text{Recall} = \frac{TP}{TP + FN} \]

- **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.
  \[ \text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \]
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.

\[ F1 \text{ Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \]

Diagnostic odds ratio (DOR): It is the ratio of the positive likelihood ratio (LR+) to the negative likelihood ratio (LR-).

\[ \text{DOR} = \frac{\text{LR}^+}{\text{LR}^-} \text{ where,} \]

Positive Likelihood Ratio (LR+): \( \frac{\text{TPR}}{\text{FPR}} \),

Negative Likelihood Ratio (LR-): \( \frac{\text{FNR}}{\text{TNR}} \),

True Positive Rate (TPR): \( \frac{\text{TP}}{\text{TP} + \text{FN}} \),

False Positive Rate (FPR): \( \frac{\text{FP}}{\text{FP} + \text{TN}} \),

False Negative Rate (FNR): \( \frac{\text{FN}}{\text{FN} + \text{TP}} \) and

True Negative Rate (TNR): \( \frac{\text{TN}}{\text{TN} + \text{FP}} \)
**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.