Techniques for Static Detection of Data-Races in OpenMP Programs

Utpal Bora

PhD Scholar
Computer Science and Engineering
IIT Hyderabad, India

April 25, 2022

Acknowledgements

Doctoral Advisory Committee:

- Prof. Ramakrishna Upadrasta (Guide)
- Prof. Saurabh Joshi (Co-Guide)
- Prof. M.V. Panduranga Rao
- Prof. Sparsh Mittal
- Prof. Subrahmanyam Kalyanasundaram

Collaborators:

- Prof. Sanjay Rajopadhaye
- Santanu Das
- Pankaj Kukreja
- Himanshu Shukla
- Shraiysh Vaishay

Mentors:

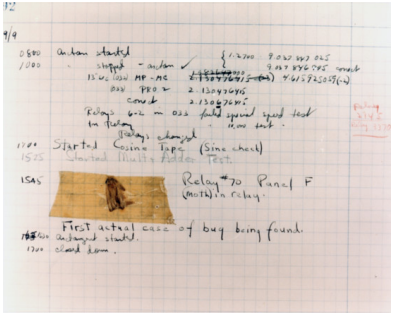
- Dr. Venugopal Raghavan
- Dr. Johannes Doerfert
- Dr. Michael Kruse
- Prof. Tobias Grosser
- Prof. Krishna Nandivada

Problem Statement

Software Bugs

"Bug" or "Glitch" or "Anomaly" or "Defect" is the incorrectness in software behaviour.

"Bug" word is popularized by Grace Hopper when she traced a problem in the Harvard Mark II system to a Moth trapped in the system.



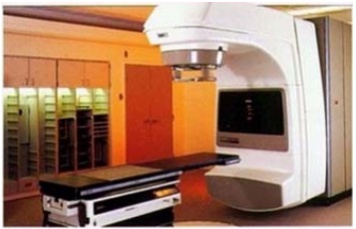
Source: *Wikipedia*

Major Incidents due to Software Bugs

Bugs can result in “Fault”/“Error” or “Failure”/“Crash”.

Major Incidents due to Software Bugs

Bugs can result in “Fault”/“Error” or “Failure”/“Crash”.



Source: *Wikipedia*

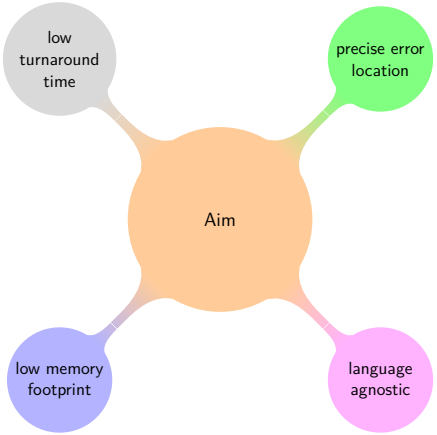
- Ariane Flight V88 Explosion
- Mariner 1 Launch Failure
- Therac-25 Radiation Therapy Accident (lives lost)
- Northeast Blackout of 2003
- Heathrow Terminal 5 Opening Blunder (500 flights cancelled)

Problem Statement

Provide a solution to the problem of statically detecting data-races in shared-memory parallel programs.

Problem Statement

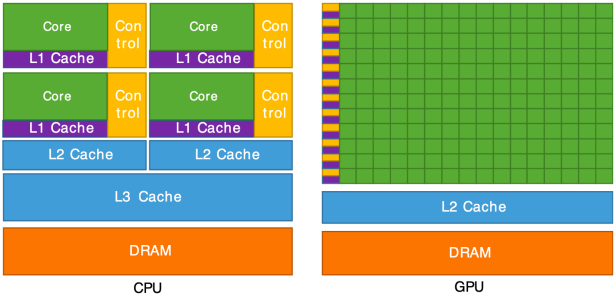
Provide a solution to the problem of statically detecting data-races in shared-memory parallel programs.



Shared-Memory Systems & Parallel Programming

Parallel Programming Models

- Shared Memory
- Message Passing
- Partitioned Global Address Space (PGAS)



Source: https://cvw.cac.cornell.edu/GPUarch/gpu_characteristics

Shared-Memory Programming Languages: Cilk, Cilk Plus, C++11, C#, CUDA, Java, OpenMP, Threading Building Blocks, etc.

OpenMP Parallel Programming

Open Multi-Processing (OpenMP) is a framework for C, C++ and FORTRAN programs to enable shared-memory parallel programming on a multi-threaded system.

OpenMP **Supports:**

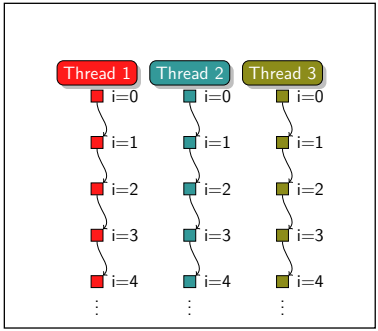
- *Single Program Multiple Data* (SMPD) programming model
- *Single Instruction Multiple Data* (SIMD) programming model
- Divide-and-conquer paradigm with tasks
- Data environment for shared-memory consistency
- Mutual exclusion and atomicity
- Implicit and explicit synchronization

OpenMP is portable across architectures from different vendors. (e.g. CPU, GPU, and FPGA [Mayer et al., 2019])

OpenMP Execution Model: parallel construct

```
1 #pragma omp parallel \
2             num_threads(3)
3 {
4   for (int i = 0; i < 15; i++) {
5     printf("Iteration Id %d by \
6            Thread Id: %d\n", i,
7            omp_get_thread_num());
8   }
9 }
```

OpenMP parallel construct

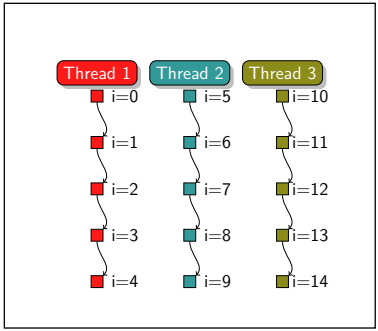


OpenMP thread execution model for parallel construct

OpenMP Execution Model: `parallel for` construct

```
1 #pragma omp parallel for \
2             schedule(static) \
3             num_threads(3)
4   for (int i = 0; i < 15 ; i++) {
5     printf("Iteration Id %d by \
6            Thread Id: %d\n", i,
7            omp_get_thread_num());
8   }
```

OpenMP worksharing `for` construct



OpenMP thread execution model for `parallel for` construct

What is a Data-Race?

```
1 #pragma omp parallel \
2             shared(b, error)
3   {
4 #pragma omp for  nowait
5     for(i = 0; i < len; i++)
6       a[i]  = b + a[i] * 5;
7 #pragma omp single
8     error =  a[9]  + 1;
9   }
```

DRB013: Example with data-race due to improper synchronization

Definition (Data-Race)

An execution of a concurrent program is said to have a data-race when

- two threads access the same memory location,
- accesses are not protected by a mutual exclusion mechanism (e.g., locks),
- the order of the two accesses is non-deterministic,
- one of these accesses is a write.

Types of data-races in OpenMP programs

- Missing data sharing clauses

```
1 #pragma omp parallel for \
2                 private (temp,i,j)
3   for (i = 0; i < len; i++)
4     for (j = 0; j < len; j++){
5       temp = u[i][j];
6       sum = sum + temp * temp;
7     }
```

DRB021: OpenMP Worksharing construct with data-race

Types of data-races in OpenMP programs

- Missing data sharing clauses
- Loop carried dependences

```
1  for (i = 0; i < n; i++) {
2 #pragma omp parallel for
3   for (j = 1; j < m; j++) {
4       b[i][j] = b[i][j-1];
5   }
6  }
```

DRB038: Example with Loop Carried Dependence

Types of data-races in OpenMP programs

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

```
1 #pragma omp simd
2 for (int i = 0; i < len-1; i++){
3   a[i+1] = a[i] + b[i];
4 }
```

DRB024: Example with SIMD data-race

Types of data-races in OpenMP programs

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

```
1 #pragma omp parallel \
2             shared(b, error)
3   {
4 #pragma omp for nowait
5     for(i = 0; i < len; i++)
6       a[i] = b + a[i] * 5;
7 #pragma omp single
8     error = a[9] + 1;
9   }
```

DRB013: Example with data-race due to improper synchronization

Types of data-races in OpenMP programs

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

```
1 #pragma omp parallel \
2             shared(Flag)
3   {
4     if (omp_get_thread_num() % 2 == 0) {
5       Flag = true;
6     }
7     //
8   }
```

Control flow dependent on number of threads

Proposed Solutions for Race Detection

- Using Polyhedral Dependence Analysis
- Using May-Happen-in-Parallel (MHP) Analysis

Race Detection using Polyhedral Dependence Analysis

Polyhedral Representation

```
1 for (i = 0; i < m;  i++) {
2 #pragma omp parallel for
3    for (j = 1; j < n;  j++) {
4 S0:   b[i][j] = b[i][j-1];
5    }
6  }
```

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

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

Polyhedral Representation

```
1 for (i = 0; i < m;  i++) {
2 #pragma omp parallel for
3    for (j = 1; j < n;  j++) {
4 S0:   b[i][j] = b[i][j-1];
5    }
6  }
```

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

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

Polyhedral Representation

```
1 for (i = 0; i < m;  i++) {
2 #pragma omp parallel for
3     for (j = 1; j < n;  j++) {
4 S0:    b[i][j] = b[i][j-1];
5     }
6  }
```

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

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

Polyhedral Representation

```
1 for (i = 0; i < m;  i++) {
2 #pragma omp parallel for
3    for (j = 1; j < n;  j++) {
4 S0:   b[i][j] = b[i][j-1];
5    }
6  }
```

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

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

Polyhedral Representation

```
1 for (i = 0; i < m;  i++) {
2 #pragma omp parallel for
3    for (j = 1; j < n;  j++) {
4 S0:   b[i][j] = b[i][j-1];
5    }
6  }
```

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

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

Polyhedral Representation

```
1 for (i = 0; i < m;  i++) {
2 #pragma omp parallel for
3    for (j = 1; j < n;  j++) {
4 S0:   b[i][j] = b[i][j-1];
5    }
6  }
```

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

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

Polyhedral Representation

```
1 for (i = 0; i < m;  i++) {
2 #pragma omp parallel for
3    for (j = 1; j < n;  j++) {
4 S0:   b[i][j] = b[i][j-1];
5    }
6  }
```

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

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

Polyhedral Representation

```
1 for (i = 0; i < m;  i++) {
2 #pragma omp parallel for
3    for (j = 1; j < n;  j++) {
4 S0:   b[i][j] = b[i][j-1];
5    }
6  }
```

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

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

Polyhedral Representation

```
1 for (i = 0; i < m;  i++) {
2 #pragma omp parallel for
3    for (j = 1; j < n;  j++) {
4 S0:   b[i][j] = b[i][j-1];
5    }
6  }
```

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

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

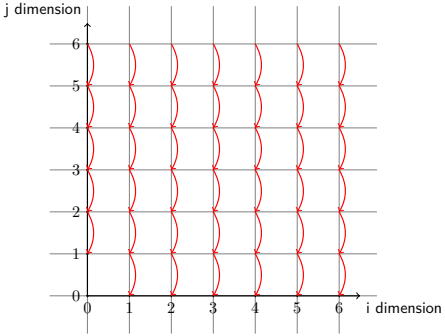
Access Map : $\mathbf{A} = \{\texttt{S0}(i,j) \rightarrow \texttt{M}(i,j); \texttt{S0}(i,j) \rightarrow \texttt{M}(i,j-1)\}$

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

Technique 1: Race Detection using Polyhedral Dependences

```
1  for (int i = 0; i < m; i++) {
2 #pragma omp parallel for
3    for (int j = 1; j < n; j++) {
4      b[i][j] = b[i][j-1];
5    }
6  }
```

Example with Loop Carried Dependence

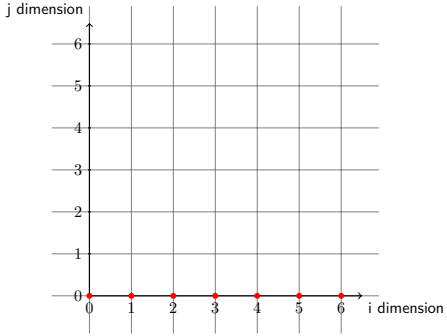


Dependence Polyhedra

Technique 1: Race Detection using Polyhedral Dependences

```
1  for (int i = 0; i < m; i++) {
2 #pragma omp parallel for
3    for (int j = 1; j < n; j++) {
4      b[i][j] = b[i][j-1];
5    }
6  }
```

Example with Loop Carried Dependence



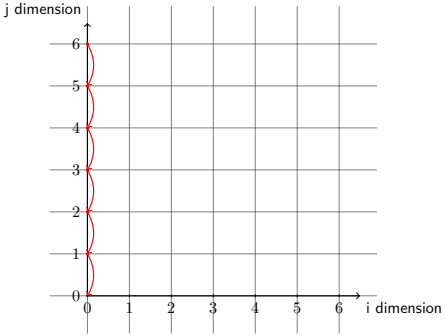
Projection of the Dependence Polyhedra on i-dimension

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

Technique 1: Race Detection using Polyhedral Dependences

```
1  for (int i = 0; i < m; i++) {
2 #pragma omp parallel for
3    for (int j = 1; j < n; j++) {
4      b[i][j] = b[i][j-1];
5    }
6  }
```

Example with Loop Carried Dependence



Projection of the Dependence Polyhedra on j-dimension

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

Overcoming Limitations of the Polyhedral Framework

- Limited scope of the Polyhedral framework (less coverage).

```
1  void f1(int* q) {
2    *q += 1;
3    // None-affine code
4  }
5
6  int main() {
7    int i = 0;
8  #pragma omp parallel
9    { f1(&i); }
10 }
```

Non-affine program

Overcoming Limitations of the Polyhedral Framework

- Limited scope of the Polyhedral framework (less coverage).
- Cannot handle inter SCoP (Static Control Parts) dependences.

```
1  void f1(int* q) {
2    *q += 1 ;
3    // None-affine code
4  }
5
6  int main() {
7    int i = 0;
8  #pragma omp parallel
9    { f1(&i); }
10 }
```

Non-affine program

```
1  #pragma omp parallel \
2                shared(b, error)
3    {
4  #pragma omp for  nowait
5      for(i = 0; i < len; i++)
6        a[i]  = b + a[i]*5;
7  #pragma omp single
8      error =  a[9]  + 1;
9    }
```

DRB013: Example with data-race due to improper synchronization

Overcoming Limitations of the Polyhedral Framework

- Limited scope of the Polyhedral framework (less coverage). Use Dependence Testing
- Cannot handle inter SCoP (Static Control Parts) dependences. Use MHP Analysis

```
1  void f1(int* q) {
2    *q += 1 ;
3    // None-affine code
4  }
5
6  int main() {
7    int i = 0;
8  #pragma omp parallel
9    { f1(&i); }
10 }
```

Non-affine program

```
1  #pragma omp parallel \
2                shared(b, error)
3    {
4  #pragma omp for  nowait
5      for(i = 0; i < len; i++)
6        a[i]  = b + a[i]*5;
7  #pragma omp single
8      error =  a[9]  + 1;
9    }
```

DRB013: Example with data-race due to improper synchronization

Race Detection using May-Happen-in-Parallel Analysis

Technique 2: Overview

- Perform Phase Interval Analysis (PIA) in the Monotone framework [Kildall, 1973]
  - Construct a TaskGraph for the OpenMP constructs
  - Compute Phase Intervals ($[lb, ub]$) for each basic block
- Perform May-Happen-in-Parallel (MHP) analysis using the Phase Intervals
  - Compute MHP sets for each basic block
- Use MHP sets to detect data-races statically

Modeling OpenMP `single` construct as TaskGraph

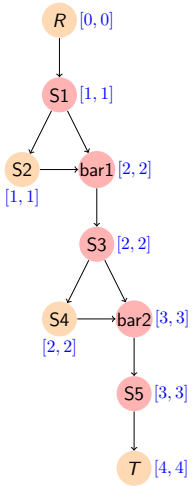
```
1  #pragma omp parallel
2    {
3      // S1
4  #pragma omp single
5      {
6        // S2
7      } // implicit barrier
8      // S3
9  #pragma omp single
10     {
11       // S4
12     } // implicit barrier
13     // S5
14   }
```

Multiple OpenMP `single` constructs

Modeling OpenMP `single` construct as TaskGraph

```
1  #pragma omp parallel
2    {
3      // S1
4  #pragma omp single
5      {
6        // S2
7      } // implicit barrier
8      // S3
9  #pragma omp single
10     {
11       // S4
12     } // implicit barrier
13     // S5
14   }
```

Multiple OpenMP `single` constructs



TaskGraph for the OpenMP `single` construct annotated with phase intervals.

Modeling OpenMP `master` construct as TaskGraph

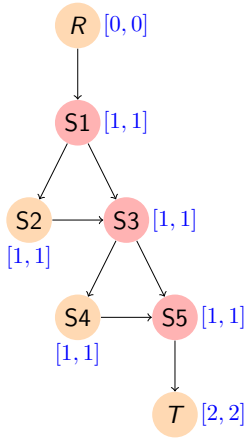
```
1  #pragma omp parallel
2    {
3      // S1
4  #pragma omp master
5      {
6        // S2
7      }
8      // S3
9  #pragma omp master
10     {
11       // S4
12     }
13     // S5
14   }
```

Multiple OpenMP `master` constructs

Modeling OpenMP `master` construct as TaskGraph

```
1  #pragma omp parallel
2    {
3      // S1
4  #pragma omp master
5      {
6        // S2
7      }
8      // S3
9  #pragma omp master
10     {
11       // S4
12     }
13     // S5
14   }
```

Multiple OpenMP `master` constructs



TaskGraph for the OpenMP `master` construct annotated with phase intervals.

Definition (Reduced TaskGraph)

A reduced TaskGraph $G = \langle V, E, R, T \rangle$ consists of:

- the vertex set V of implicit and explicit tasks.
- the edge set $E \subseteq V \times V$ contains directed edges.
- a special vertex R for the root of the graph.
- a special sentinel vertex T that represents the termination of all the tasks.

Definition (Phase)

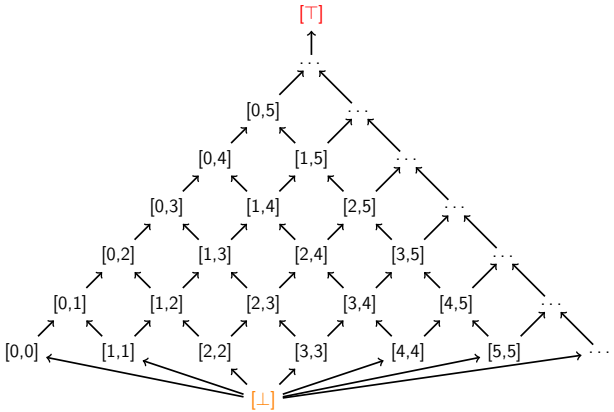
A phase, represented by a positive integer $p \in \mathbb{N}^+ \cup \infty$, is the execution number of an instance of the nodes in the TaskGraph.

A program can execute in phases due to:

- the presence of `barrier`s
- upon entry to a `parallel` region
- upon exit from a `parallel` region.

Formalizing Phase Interval Analysis (PIA)

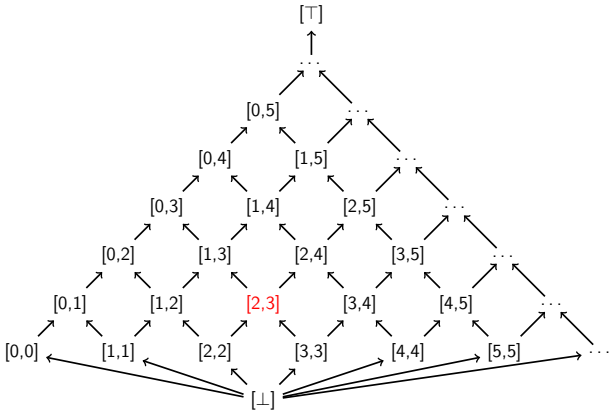
Lattice : $(I, \sqsubseteq, \sqcup, \sqcap)$ is a lattice over an interval domain I.
Top element: $\top = [0, \infty]$. Bot element: $\bot$.



Interval Lattice for PIA

Formalizing Phase Interval Analysis (PIA)

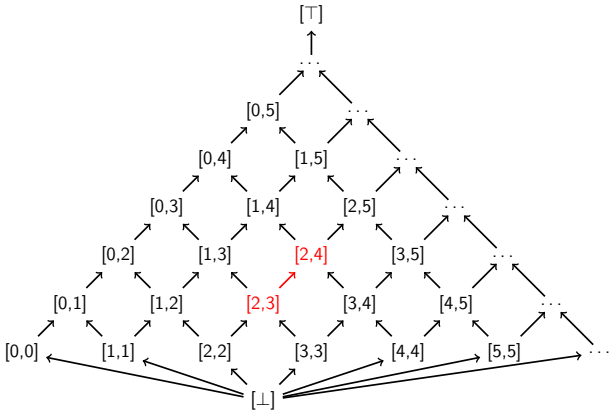
Lattice : $(I, \sqsubseteq, \sqcup, \sqcap)$ is a lattice over an interval domain I.
Interval (PI): $[lb, ub] \in I$, where lb is lower bound & ub is upper bound.



Interval Lattice for PIA

Formalizing Phase Interval Analysis (PIA)

Lattice : $(I, \sqsubseteq, \sqcup, \sqcap)$ is a lattice over an interval domain I.
Partial Order ($\sqsubseteq$): $[lb_1, ub_1] \sqsubseteq [lb_2, ub_2] \iff lb_2 \leq lb_1 \leq ub_1 \leq ub_2$.

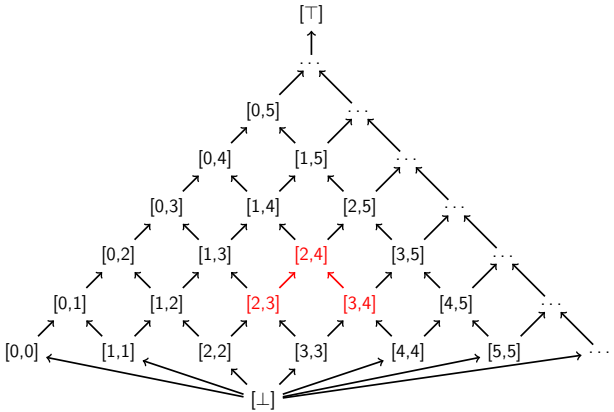


Interval Lattice for PIA

Formalizing Phase Interval Analysis (PIA)

Lattice : $(I, \sqsubseteq, \sqcup, \sqcap)$ is a lattice over an interval domain I.

Join Operation ($\sqcup$): $[lb_1, ub_1] \sqcup [lb_2, ub_2] \implies [\min(lb_1, lb_2), \max(ub_1, ub_2)]$.

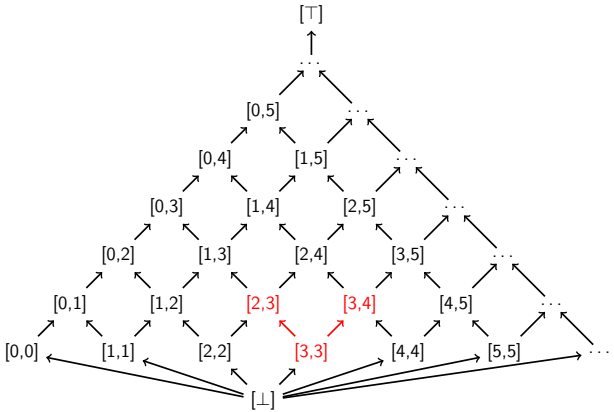


Interval Lattice for PIA

Formalizing Phase Interval Analysis (PIA)

Lattice : $(I, \sqsubseteq, \sqcup, \sqcap)$ is a lattice over an interval domain I.

Meet Operation ($\sqcap$): $[lb_1, ub_1] \sqcap [lb_2, ub_2] \implies [\max(lb_1, lb_2), \min(ub_1, ub_2)]$.



Interval Lattice for PIA

Phase Interval Analysis

Data flow Equation for Phase Interval Analysis for a node $S \in G$ is defined as:

$$PI_{in}[S] = \bigsqcup_{p \in pred[S]} PI_{out}[p]$$

where the confluence operator is the join operation ($\sqcup$), p is an immediate predecessor node of S, and $pred[S]$ is the set of all the immediate predecessor nodes of $S \in G$.

Data flow Equation for Phase Interval Analysis for a node $S \in G$ is defined as:

$$PI_{in}[S] = \bigsqcup_{p \in pred[S]} PI_{out}[p]$$

where the confluence operator is the join operation ($\sqcup$), p is an immediate predecessor node of S, and $pred[S]$ is the set of all the immediate predecessor nodes of $S \in G$.

Transfer Function is defined as:

$$PI_{out}(S) = f(PI_{in}(S)) = \begin{cases} PI_{in}(S), & \text{Identity function, if } S \\ & \text{does not have a } \texttt{barrier}. \\ PI_{in}(S) + [1,1], & \text{otherwise.} \end{cases}$$

The binary operator $+$ is defined as $[lb_1, ub_1] + [lb_2, ub_2] = [lb_1 + lb_2, ub_1 + ub_2]$.

Phase Interval Analysis: Termination Guarantee

Widening is defined as:

$$[lb_1, ub_1]\nabla_{[lbt,ubt]}[lb_2, ub_2] = \begin{cases} \begin{cases} [lbt, ubt] & \text{if } ub_1 < ub_2 \\ [lbt, ub_1] & \text{otherwise} \end{cases} & \text{if } lb_1 < lb_2 \\ \begin{cases} [lb_1, ubt] & \text{if } ub_1 < ub_2 \\ [lb_1, ub_1] & \text{otherwise} \end{cases} & \text{otherwise} \end{cases}$$

where *lbt* and *ubt* are widening thresholds.

Phase Interval Analysis: Termination Guarantee

Widening is defined as:

$$[lb_1, ub_1]\nabla_{[lbt,ubt]}[lb_2, ub_2] = \begin{cases} \begin{cases} [lbt, ubt] & \text{if } ub_1 < ub_2 \\ [lbt, ub_1] & \text{otherwise} \end{cases} & \text{if } lb_1 < lb_2 \\ \begin{cases} [lb_1, ubt] & \text{if } ub_1 < ub_2 \\ [lb_1, ub_1] & \text{otherwise} \end{cases} & \text{otherwise} \end{cases}$$

where lbt and ubt are widening thresholds.

For a loop header H and loop trip count TC, widening is applied as:

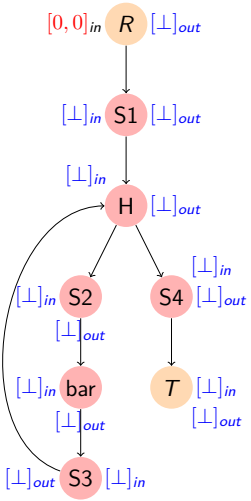
$$PI_{in}(H) = \begin{cases} PI^0_{in}(H) \sqcup (PI^0_{in}(H) + (TC * (PI^0_{in}(H) \Delta PI^1_{in}(H)))) & \text{If the trip count } TC \text{ is known} \\ PI^0_{in}(H) \nabla_{[0,\infty]} PI^1_{in}(H) & \text{otherwise} \end{cases}$$

where difference (Δ): $[lb_1, ub_1]\Delta[lb_2, ub_2] \implies [lb_2 - lb_1, ub_2 - ub_1]$.

Phase Interval Analysis: Working Example

```
1  #pragma omp parallel
2    {
3      // S1
4      for (int i = 0; i < 100; i++) {
5        // S2
6  #pragma omp barrier
7        // S3
8      }
9      // S4
10   }
```

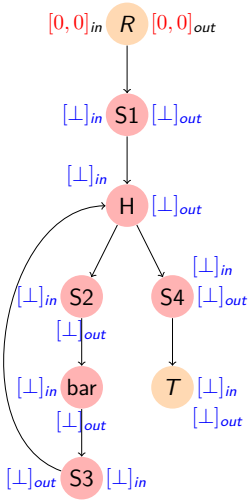
OpenMP `parallel` construct



Phase Interval Analysis: Working Example

```
1  #pragma omp parallel
2    {
3      // S1
4      for (int i = 0; i < 100; i++) {
5        // S2
6  #pragma omp barrier
7        // S3
8      }
9      // S4
10   }
```

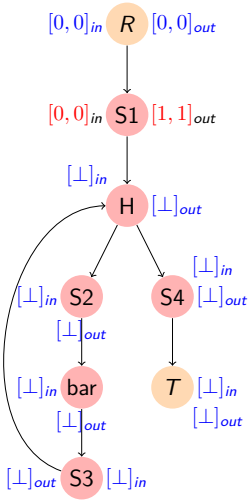
OpenMP `parallel` construct



Phase Interval Analysis: Working Example

```
1  #pragma omp parallel
2    {
3      // S1
4      for (int i = 0; i < 100; i++) {
5        // S2
6  #pragma omp barrier
7        // S3
8      }
9      // S4
10   }
```

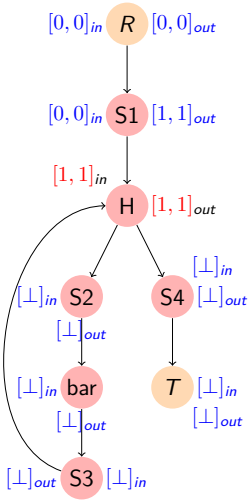
OpenMP `parallel` construct



Phase Interval Analysis: Working Example

```
1  #pragma omp parallel
2    {
3      // S1
4      for (int i = 0; i < 100; i++) {
5        // S2
6  #pragma omp barrier
7        // S3
8      }
9      // S4
10   }
```

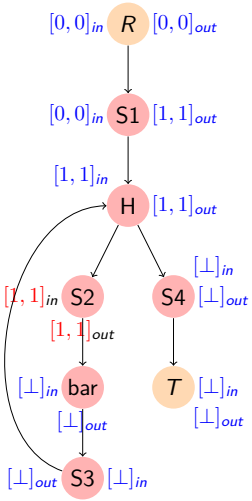
OpenMP `parallel` construct



Phase Interval Analysis: Working Example

```
1  #pragma omp parallel
2    {
3      // S1
4      for (int i = 0; i < 100; i++) {
5        // S2
6  #pragma omp barrier
7        // S3
8      }
9      // S4
10   }
```

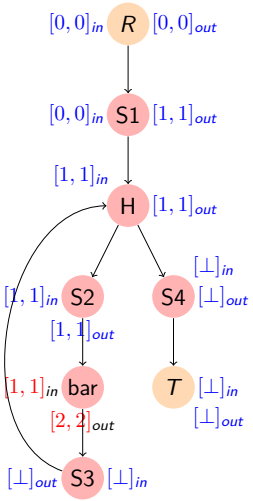
OpenMP `parallel` construct



Phase Interval Analysis: Working Example

```
1  #pragma omp parallel
2    {
3      // S1
4      for (int i = 0; i < 100; i++) {
5        // S2
6  #pragma omp barrier
7        // S3
8      }
9      // S4
10   }
```

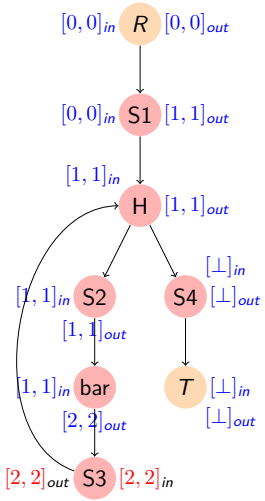
OpenMP `parallel` construct



Phase Interval Analysis: Working Example

```
1  #pragma omp parallel
2    {
3      // S1
4      for (int i = 0; i < 100; i++) {
5        // S2
6  #pragma omp barrier
7        // S3
8      }
9      // S4
10   }
```

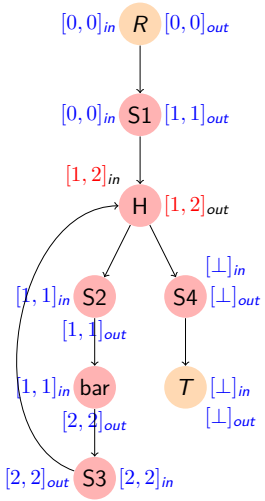
OpenMP `parallel` construct



Phase Interval Analysis: Working Example

```
1  #pragma omp parallel
2    {
3      // S1
4      for (int i = 0; i < 100; i++) {
5        // S2
6  #pragma omp barrier
7        // S3
8      }
9      // S4
10   }
```

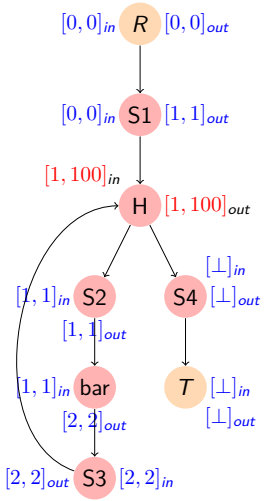
OpenMP `parallel` construct



Phase Interval Analysis: Working Example

```
1  #pragma omp parallel
2    {
3      // S1
4      for (int i = 0; i < 100; i++) {
5        // S2
6  #pragma omp barrier
7        // S3
8      }
9      // S4
10   }
```

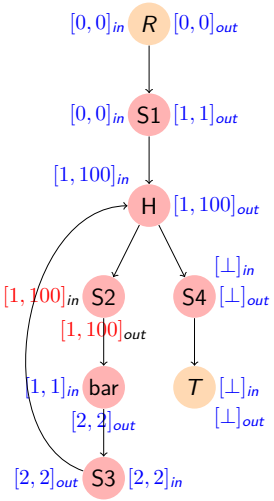
OpenMP `parallel` construct



Phase Interval Analysis: Working Example

```
1  #pragma omp parallel
2    {
3      // S1
4      for (int i = 0; i < 100; i++) {
5        // S2
6  #pragma omp barrier
7        // S3
8      }
9      // S4
10   }
```

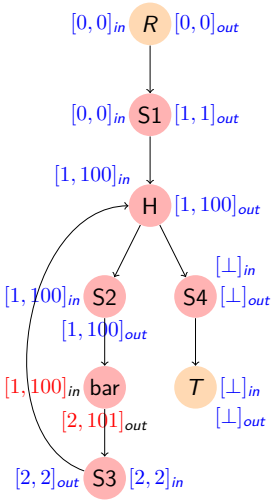
OpenMP `parallel` construct



Phase Interval Analysis: Working Example

```
1  #pragma omp parallel
2    {
3      // S1
4      for (int i = 0; i < 100; i++) {
5        // S2
6  #pragma omp barrier
7        // S3
8      }
9      // S4
10   }
```

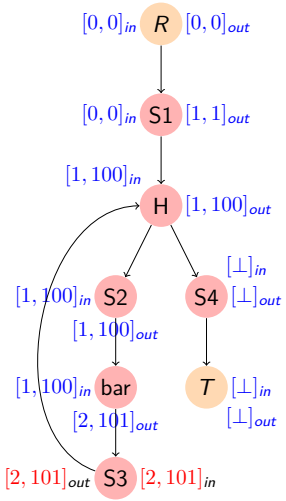
OpenMP `parallel` construct



Phase Interval Analysis: Working Example

```
1  #pragma omp parallel
2    {
3      // S1
4      for (int i = 0; i < 100; i++) {
5        // S2
6  #pragma omp barrier
7        // S3
8      }
9      // S4
10   }
```

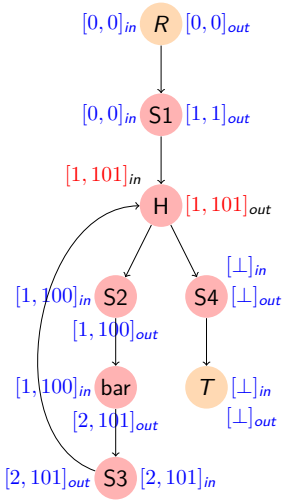
OpenMP `parallel` construct



Phase Interval Analysis: Working Example

```
1  #pragma omp parallel
2    {
3      // S1
4      for (int i = 0; i < 100; i++) {
5        // S2
6  #pragma omp barrier
7        // S3
8      }
9      // S4
10   }
```

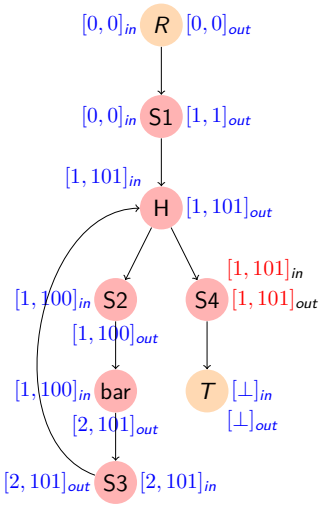
OpenMP `parallel` construct



Phase Interval Analysis: Working Example

```
1  #pragma omp parallel
2    {
3      // S1
4      for (int i = 0; i < 100; i++) {
5        // S2
6  #pragma omp barrier
7        // S3
8      }
9      // S4
10   }
```

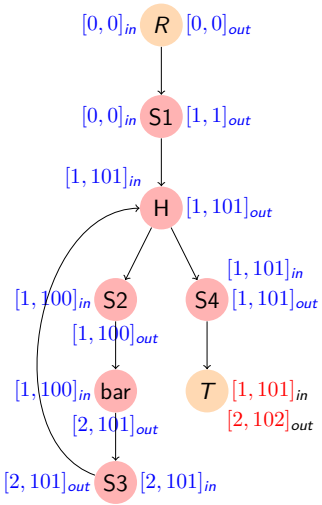
OpenMP `parallel` construct



Phase Interval Analysis: Working Example

```
1  #pragma omp parallel
2    {
3      // S1
4      for (int i = 0; i < 100; i++) {
5        // S2
6  #pragma omp barrier
7        // S3
8      }
9      // S4
10   }
```

OpenMP `parallel` construct



Technique 2: May-Happen-in-Parallel (MHP) Analysis

Given: TaskGraph and Phase Intervals.

Question: Can the two statements S1 and S2 be executed in parallel?

```
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; //S1
6 #pragma omp single
7     error = a[9] + 1;  //S2
8   }
```

Technique 2: May-Happen-in-Parallel (MHP) Analysis

Given: TaskGraph and Phase Intervals.
Question: Can the two statements S1 and S2 be executed in parallel?

```
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; //S1
6 #pragma omp single
7     error = a[9] + 1;  //S2
8   }
```

Answer: Instances of two nodes $u, v \in G$ may run in parallel if and only if $PI[u] \bigcap PI[v]$. Here the binary operator $\bigcap$ on the phase intervals $PI_1 = [lb_1, ub_1]$ and $PI_2 = [lb_2, ub_2]$ is:

$$PI_1 \bigcap PI_2 = \begin{cases} true, & lb_1 \leq ub_2 \leq ub_1 \vee \\ & lb_2 \leq ub_1 \leq ub_2 \\ false, & \text{otherwise.} \end{cases}$$

Implementation Details

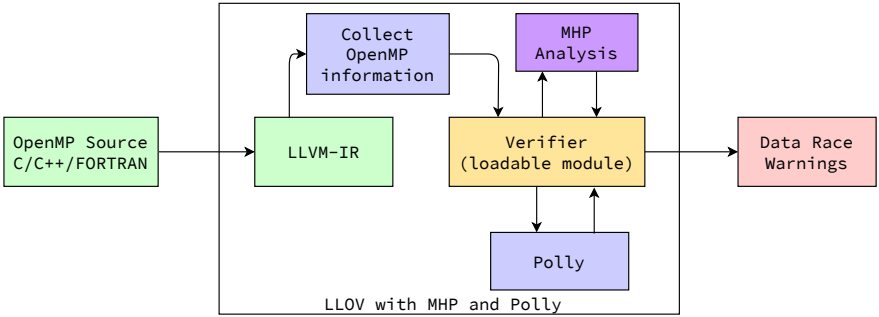
Implementation Details

Designed and implemented a static data-race checker, LLOV, in the LLVM compiler framework.

LLOV:

- is based on the intermediate representation of LLVM (LLVM-IR)
- uses polyhedral dependence analysis
- uses May-Happen-in-Parallel analysis
- can handle OpenMP programs in C/C++/FORTRAN
- also can conservatively state when a program is data-race free

LLOV: Architecture



Flow Diagram of LLVM OpenMP Verifier (LLOV)

LLOV: Reconstructing pragmas from IR

```
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   }
```

LLOV: Reconstructing pragmas from IR

```
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   }
```

```
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
```

In-memory representation of a directive.

LLOV: Reconstructing pragmas from IR

```
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   }
```

```
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
```

In-memory representation of a directive.

```
<Directive> ::= <Dtype> [ <Sched> ] { <Var> } { <Directive> }
<Dtype>     ::= parallel | for | simd
                | workshare | sections
                | section | single | master
                | critical | atomic
                | target | distribute
<Var>       ::= <Vtype> val
<Vtype>     ::= private | firstprivate
                | lastprivate | threadprivate
                | shared | reduction
<Sched>     ::= [ <modifier> ] [ ordered ] <Stype> <chunk>
<modifier>  ::= monotonic | nonmonotonic
<Stype>     ::= static | dynamic
                | guided | auto | runtime
<chunk>     ::= <non-negative integer>
```

Data-Race Reporting in LLOV (Goal 1)

Data-Race: The source memory access MA_1 (in basic block u) and sink memory access MA_2 (in basic block v) of a data dependence may potentially race if $PI[u] \bigcap PI[v]$.

```
Data Race detected.
Source : llvm/lib/Transforms/OpenMPVerify/test/10.race1.c:10:11
Sink : llvm/lib/Transforms/OpenMPVerify/test/10.race1.c:12:11
==============
9 :     {
10 :   x = 1;
11 : #pragma omp section
12 :   x = 2;
13 :    }
==============
```

Error reporting: LLOV highlights the source location of the race condition.

Experimental Evaluation

Evaluation

We evaluate our work on two metrics:

- Compare Performance with seeded benchmarks
- Measure Scalability with real world applications

Experimental Setup

Benchmarks

- DataRaceBench C/C++ v1.3.2 [Verma et al., 2020] to evaluate performance
  - 172 Kernels, 83 kernels with data-races, 89 kernels without any data-race
- 22 ECP Proxy application to evaluate scalability (1 million LOC)
- DataRaceBench FORTRAN [Kukreja et al., 2019]

Experimental Setup

Benchmarks

- DataRaceBench C/C++ v1.3.2 [Verma et al., 2020] to evaluate performance
  - 172 Kernels, 83 kernels with data-races, 89 kernels without any data-race
- 22 ECP Proxy application to evaluate scalability (1 million LOC)
- DataRaceBench FORTRAN [Kukreja et al., 2019]

System Specifications

System: Two Intel Xeon E5-2697 v4 @ 2.30GHz processors
OS: 64 bit Ubuntu 20.04.2 LTS Server
Kernel: Linux kernel version 5.4.0-71-generic
Threads: 72 (2 x 36) hardware threads
Memory: 128GB
OpenMP library: LLVM OpenMP runtime v5.0.1 (libomp5)

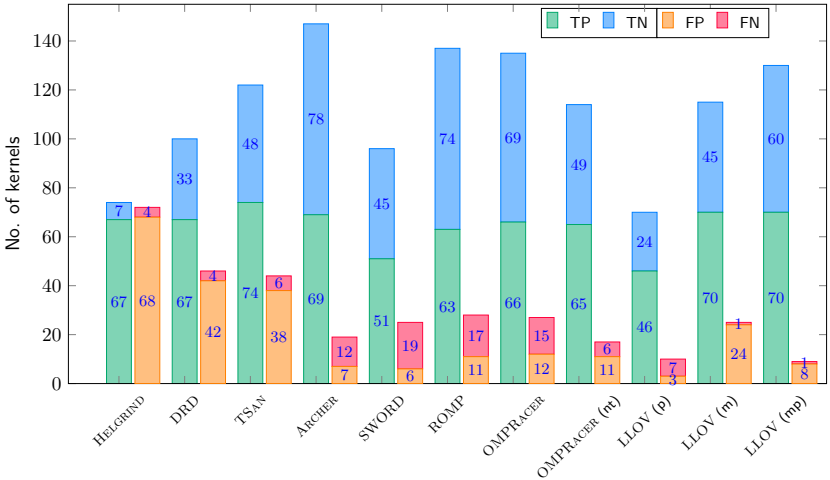
Other Race Detection Tools

Race detection tools with the version numbers and flags used for comparison. Dynamic tool, Static tool.

Tools	Version	Flags	OpenMP	Compiler
HELGRIND [Valgrind-project, 2007b]	3.15.0	*–tool=helgrind*	gomp	gcc 9.3
VALGRIND DRD [Valgrind-project, 2007a]	3.15.0	*–tool=drd –check-stack-var=yes*	gomp	gcc 9.3
TSAN-LLVM [Serebryany et al., 2011]	12.0.1	*ignore_noninstrumented_modules=1*	lomp	clang 12.0
ARCHER [Atzeni et al., 2016]	5b37681	*ignore_noninstrumented_modules=1*	lomp	clang 6.0
SWORD [Atzeni et al., 2018]	7a08f3c	*–analysis-tool=sword-race-analysis*	lomp	clang 7.1
ROMP [Gu and Mellor-Crummey, 2018]	v2.0		gomp	gcc 9.3
OMPRACER [Swain et al., 2020]	0.1.1	*–no-filter –silent –nolimit*	lomp	clang 9.0
LLOV (WITH POLLY)	v0.3	*openmp-verify-use-llvm-da=false*	lomp	clang 12.0
LLOV (WITH MHP)	v0.3	*openmp-verify-use-polly-da=false*	lomp	clang 12.0
LLOV (WITH MHP & POLLY)	v0.3	*-Xclang -disable-O0-optnone*	lomp	clang 12.0

Unavailable tools: OMPVERIFY [Basupalli et al., 2011], POLYOMP [Chatarasi et al., 2016], and DRACO [Ye et al., 2018].

Performance: DataRaceBench v1.3.2 Results



Tools (nt for without tasks, p for Polly, m for MHP, mp for MHP and Polly)

Maximum number of races reported by different tools in DataRaceBench v1.3.2

Performance: DataRaceBench v1.3.2 Statistics

Precision, Recall, and Accuracy of the tools on DataRaceBench v1.3.2

Tools	Precision	Recall	Accuracy	F1 Score	Diagnostic Odds Ratio
Helgrind	0.50	**0.94**	0.51	0.65	1.72
Valgrind DRD	0.61	**0.94**	0.68	0.74	13.16
TSan-LLVM	0.66	0.93	0.73	0.77	15.58
Archer	**0.91**	0.85	**0.89**	0.88	64.07
SWORD	0.89	0.73	0.79	0.80	20.13
ROMP	0.85	0.79	0.83	0.82	24.93
OMPRacer	0.85	0.81	0.83	0.83	25.30
OMPRacer (notasks)	0.86	0.92	0.87	0.88	48.26
LLOV (with Polly)	**0.94**	0.87	0.88	**0.90**	52.57
LLOV (with MHP)	0.74	**0.99**	0.82	0.85	**131.25**
LLOV (with MHP & Polly)	0.90	**0.99**	**0.94**	**0.94**	**525.00**

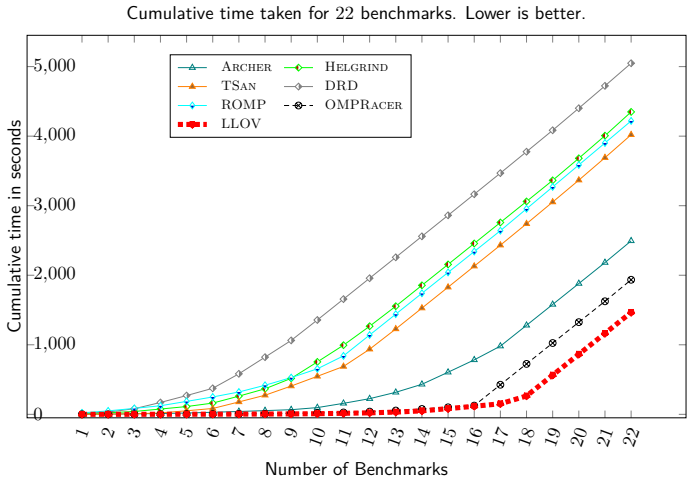
Scalability Analysis on million LOC

22 Miscellaneous Benchmarks used for Scalability Analysis

Benchmark	Domain	KLOC	Benchmark	Domain	KLOC
AMG	Parallel Algebraic Multigrid Solver	64.9	NBody	Parallel N-body Simulation	0.4
CoMD	Molecular Dynamics Simulation	6.1	openmp-tutorial	SC OpenMP Tutorial	10.4
COVID-SIM	COVID-19 Transmission Dynamics	19.0	OpenMP-Microbench	OpenMP Performance Benchmarking	1.0
LCALS	Livermore Loops	6.6	ParallelResearchKernels	Parallel Architectures	106.2
LULESH	Shock Hydrodynamics	5.5	QuickSilver	Monte Carlo Particle Transport	10.0
miniAMR	Adaptive Mesh Refinement	17.7	Rodinia	Parallel Computing Benchmark Suite	186.4
miniBUDE	Molecular Docking	9.5	RSBench	Monte Carlo Neutron Transport	6.0
miniFE	Finite Element	297.8	sw4lite	Geodynamics	54.2
miniQMC	Quantum Monte Carlo Simulation	31.9	Sundials	Nonlinear ODE Solver	186.9
miniVite	Louvain Method for Community Detection	2.4	TriangleCounting	Parallel Triangle Counting	0.8
NAS-Parallel	Performance of Parallel Supercomputers	15.9	XSBench	Monte Carlo Neutron Transport	6.7

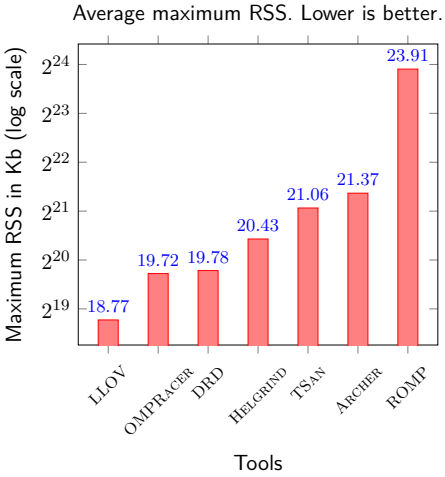
Combined total of 1 million lines of code.

Scalability Analysis: Runtime (Goal 2)



Cactus plot of different race detection tools on 22 miscellaneous benchmarks.

Scalability Analysis: Memory Footprint (Goal 3)



Average maximum resident set size (RSS) of the tools across all the 22 miscellaneous benchmarks

DataRaceBench FORTRAN (Goal 4)

An implementation of DataRaceBench C/C++ v1.2 [Liao et al., 2017] in FORTRAN 95.

- Demonstrate LLOV is language agnostic
- Created DataRaceBench FORTRAN
  - Converted 92 (out of 116) C/C++ kernels
- Open-sourced the benchmark [Kukreja et al., 2019]

[1] Earlier version of LLOV. FORTRAN frontend (FLANG) generating LLVM-IR is replaced by a new frontend (F18) that does not yet generate LLVM-IR.

DataRaceBench FORTRAN (Goal 4)

An implementation of DataRaceBench C/C++ v1.2 [Liao et al., 2017] in FORTRAN 95.

- Demonstrate LLOV is language agnostic
- Created DataRaceBench FORTRAN
  - Converted 92 (out of 116) C/C++ kernels
- Open-sourced the benchmark [Kukreja et al., 2019]

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 v0.2[1]	36	7	19	5	67

[1]Earlier version of LLOV. FORTRAN frontend (FLANG) generating LLVM-IR is replaced by a new frontend (F18) that does not yet generate LLVM-IR.

OpenMP v4.5 Pragma Handling Status

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

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

Summary

We proposed and implemented a fast, static data-race checker for shared-memory parallel programs using

- Polyhedral Dependence Analysis
- May-Happen-in-Parallel (MHP) Analysis

Summary

We proposed and implemented a fast, static data-race checker for shared-memory parallel programs using

- Polyhedral Dependence Analysis
- May-Happen-in-Parallel (MHP) Analysis

Observations

- Using only precise dependence analysis results in low coverage but high precision
- Using only MHP analysis results in more coverage but less precision

Conclusion

- Combination of precise dependence analysis and MHP analysis results in high precision, high accuracy, and better coverage

Future Directions

Handling explicit tasks in OpenMP

Support tasking constructs: `task`, `taskloop`, `taskgroup`, and `taskwait`.
None of the static techniques support tasking. (Open Problem).

MLIR based LLOV

Use OpenMP dialect of MLIR in place of LLVM-IR.

PIA for Optimization

PIA for C++ threads and `pthreds` will be interesting, and will expose classic compiler optimizations on shared-memory parallel programs.

List of Publications

- **Utpal Bora**, Santanu Das, Pankaj Kukreja, Saurabh Joshi, Ramakrishna Upadrasta, and Sanjay Rajopadhye. 2020. "LLOV: A Fast Static Data-Race Checker for OpenMP Programs." *ACM Trans. Archit. Code Optim.* 17, 4, Article 35 (December 2020), 26 pages. DOI:https://doi.org/10.1145/3418597

- **Utpal Bora**, Shraiysh Vaishay, Saurabh Joshi and Ramakrishna Upadrasta, "OpenMP aware MHP Analysis for Improved Static Data-Race Detection", *2021 IEEE/ACM 7th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC)*, 2021, pp. 1-11, doi: 10.1109/LLVMHPC54804.2021.00006

- Shalini Jain, **Utpal Bora**, Prateek Kumar, Vaibhav B. Sinha, Suresh Purini and Ramakrishna Upadrasta, "An analysis of executable size reduction by LLVM passes." *CSI Transactions on ICT* 7, 105110 (2019). doi: 10.1007/s40012-019-00248-5

Thank you for your time! Questions?

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

utpalbora.com
(ub230@cam.ac.uk)

Thank You!!

References I

Atzeni, S., Gopalakrishnan, G., Rakamarić, 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 *2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016, Chicago, IL, USA, May 23-27, 2016*, pages 53–62. IEEE Computer Society.

Atzeni, S., Gopalakrishnan, G., Rakamarić, 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 2018, Vancouver, BC, Canada, May 21-25, 2018*, pages 845–854. IEEE Computer Society.

Basupalli, V., Yuki, T., Rajopadhye, S., Morvan, A., Derrien, S., Quinton, P., and Wonnacott, D. (2011).
ompverify: Polyhedral analysis for the openmp programmer.
In Chapman, B. M., Gropp, W. D., Kumaran, K., and Müller, M. S., editors, *OpenMP in the Petascale Era - 7th International Workshop on OpenMP, IWOMP 2011, Chicago, IL, USA, June 13-15, 2011. Proceedings*, volume 6665 of *Lecture Notes in Computer Science*, pages 37–53, Berlin, Heidelberg. Springer Berlin Heidelberg.

Chatarasi, P., Shirako, J., Kong, M., and Sarkar, V. (2016).
An extended polyhedral model for SPMD programs and its use in static data race detection.
In Ding, C., Criswell, J., and Wu, P., editors, *Languages and Compilers for Parallel Computing - 29th International Workshop, LCPC 2016, Rochester, NY, USA, September 28-30, 2016, Revised Papers*, volume 10136 of *Lecture Notes in Computer Science*, pages 106–120. Springer.

Gu, Y. and Mellor-Crummey, J. (2018).
Dynamic data race detection for openmp programs.
In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '18. IEEE Press.

References II

Kildall, G. A. (1973).
A unified approach to global program optimization.
In *Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '73, page 194206, New York, NY, USA. Association for Computing Machinery.

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. (2017).
Dataracebench: A benchmark suite for systematic evaluation of data race detection tools.
In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, pages 11:1–11:14, New York, NY, USA. ACM.

Mayer, F., Knaust, M., and Philippsen, M. (2019).
Openmp on fpgas—a survey.
In Fan, X., de Supinski, B. R., Sinnen, O., and Giacaman, N., editors, *OpenMP: Conquering the Full Hardware Spectrum*, pages 94–108, Cham. Springer International Publishing.

Serebryany, K., Potapenko, A., Iskhodzhanov, T., and Vyukov, D. (2011).
Dynamic race detection with llvm compiler.
In *Proceedings of the Second International Conference on Runtime Verification*, RV'11, pages 110–114, Berlin, Heidelberg. Springer-Verlag.

Swain, B., Li, Y., Liu, P., Laguna, I., Georgakoudis, G., and Huang, J. (2020).
OMPRacer: A Scalable and Precise Static Race Detector for OpenMP Programs.
In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press.

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].

Verma, G., Shi, Y., Liao, C., Chapman, B., and Yan, Y. (2020).
Enhancing dataracebench for evaluating data race detection tools.
In *2020 IEEE/ACM 4th International Workshop on Software Correctness for HPC Applications (Correctness)*, pages 20–30.

Ye, F., Schordan, M., Liao, C., Lin, P., Karlin, I., and Sarkar, V. (2018).
Using polyhedral analysis to verify openmp applications are data race free.
In Laguna, I. and Rubio-González, C., editors, *2nd IEEE/ACM International Workshop on Software Correctness for HPC Applications, CORRECTNESS@SC 2018, Dallas, TX, USA, November 12, 2018*, pages 42–50. IEEE.

Extra Slides!

Contributions

- Implemented a *fast*, *static*, and *language agnostic* OpenMP data-race checker
- Demonstrated use of Pohyhedral dependences and MHP in data-race detection
- Proposed and Formalized OpenMP aware MHP Analysis using Phase Interval Analysis
- Created DataRaceBench FORTRAN, an annotated benchmark of data-races in FORTRAN
- Visualization of TaskGraph for better developer productivity

Data-Races in Miscellaneous Benchmarks

Number of Data-Races reported by various tools in 22 Miscellaneous Benchmarks

Benchmark	**KLOC**	LLOV	TSAN	ARCHER	OMPRACER	ROMP	HELGRIND	DRD
AMG	64.9	644	536	97	2489	9520	1000	1000
CoMD	6.1	22	71	0	12	0	367	129
COVID-SIM	19.0	331	2	0	0	0	350	0
LCALS	6.6	18	0	0	0	0	0	0
LULESH	5.5	126	71	0	139	0	1000	648
miniAMR	17.7	7693	0	250	539	146644	159	106
miniBUDE	9.5	0	1	0	0	0	131	57
miniFE	297.8	0	177	0	2	0	743	332
miniQMC	31.9	593	90	0	0	0	193	143
miniVite	2.4	5	66	0	68	14048	393	218
NAS-Parallel	15.9	219	17	84	8	0	73	10
NBody	0.4	18	99	0	4	0	133	140
openmp-tutorial	10.4	4	462	1	1	0	203	24
OpenMP-Microbench	1.0	0	46	2	0	0	335	19
ParallelResearchKernels	106.2	31	13	31	4165	36840	105	33
QuickSilver	10.0	0	48	33	7	0	128	2
Rodinia	186.4	55	166	3590	33	2231871	206	121
RSBench	6.0	0	4	0	0	0	13	0
sw4lite	54.2	42	895	0	0	0	112	112
Sundials	186.9	59	13	64	0	0	164	78
TriangleCounting	0.8	11	100	1	0	0	576	273
XSBench	6.7	0	6	0	0	0	23	0

LLOV: PIA Algorithm I

1: **Input:** TaskGraph G
2: **Output:** Phase Intervals PI
3: **Data:** worklist
4: **function** PIA(G)
5: worklist.add($G.root$)
6: **for all** n $\in$ G **do**
7: $PI_{out}[n] \leftarrow \perp$ ▷ init to bot
8: **if** n has `barrier` **then** worklist.add(n) **end if**
9: **end for**
10: $PI_{in}[G.root] \leftarrow [0, 0]$
11: **while** !worklist.empty() **do**
12: $n \leftarrow$ worklist.pop()
13: $PI_{in}[n] \leftarrow \bigsqcup_{p \in pred[n]} PI_{out}[p]$ ▷ join

LLOV: PIA Algorithm II

```
14:         if n is loop header then
15:             PI_out[n] ← w(PI_in[n])                    ▷ widen
16:         end if
17:         PI_out[n] ← f(PI_in[n])                        ▷ transfer
18:         if PI_out[n] ≠ PI_in[n] then
19:             for all s ∈ successor(n) do
20:                 worklist.add(s)
21:             end for
22:         end if
23:     end while
24:     return PI_out
25: end function
```

Data-race detection literature

Race detection literature (minimalistic)

Published	Title	Year
TOCS	Eraser: A dynamic data race detector for multithreaded programs	1997
SOSP	RacerX: Effective, Static Detection of Race Conditions and Deadlocks	2003
POPL	Conditional must not aliasing for static race detection	2007
PPoPP	May-happen-in-parallel analysis of X10 program	2007
ESEC-FSE	RELAY: static race detection on millions of lines of code	2007
WBIA	ThreadSanitizer data race detection in practice	2009
IWOMP	ompVerify : Polyhedral Analysis for the OpenMP Programmer	2011
TOPLAS	LOCKSMITH: Practical static race detection for C	2011
LCPC	An extended polyhedral model for SPMD programs and its use in static data race detection	2016
IPDPS	ARCHER: effectively spotting data races in large OpenMP applications	2016
IPDPS	SWORD: A Bounded Memory-Overhead Detector of OpenMP Data Races in Production Runs	2018
Correctness	Using Polyhedral Analysis to Verify OpenMP Applications are Data Race Free	2018
SC	Dynamic Data Race Detection for OpenMP Programs	2018
OOPSLA	RacerD: Compositional Static Race Detection	2018
IWOMP	OMPSan: Static Verification of OpenMPs Data Mapping Constructs	2019
SC	OMPRacer: A Scalable and Precise Static Race Detector for OpenMP Programs	2020
TACO	LLOV: A Fast Static Data-Race Checker for OpenMP Programs	2020

The only FN case for LLOV

```
1   #pragma omp parallel shared(a) private(i)
2   {
3     #pragma omp master
4     a = 0;
5
6     #pragma omp for reduction(+:a)
7     for (i=0; i<10; i++){
8       a = a + i;
9     }
10
11    #pragma omp single
12    printf("Sum is %d\n", a);
13  }
```

DRB140: Example of the only FN case for LLOV

Improvement area: PIA can be made more precise

```
1     #pragma omp parallel
2        for(int i=0; i<N ;i++){
3         #pragma omp for
4         for(int i=0; i<C; i++){
5           temp[i] = b[i] + c[i];
6         }
7
8         #pragma omp for
9         for(int i=C-1; i>=0; i--){
10          b[i] = temp[i] * a;
11        }
12      }
```

DRB159: PIA is not precise due to conservative analysis

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.
  $Precision = \frac{TP}{TP + FP}$
- **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.