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

December 17, 2020

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 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* (SPMD) programming model

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* (SPMD) programming model
- *Single Instruction Multiple Data* (SIMD) programming model

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* (SPMD) programming model
- *Single Instruction Multiple Data* (SIMD) programming model
- Divide-and-conquer paradigm with tasks

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* (SPMD) programming model
- *Single Instruction Multiple Data* (SIMD) programming model
- Divide-and-conquer paradigm with tasks
- Data environment for shared memory consistency

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* (SPMD) 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

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* (SPMD) 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 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* (SPMD) 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 Example: #pragma omp parallel

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

Listing 1: OpenMP parallel construct

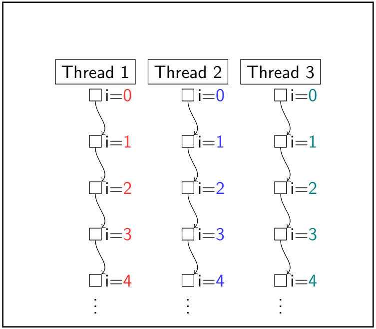

Figure 1: OpenMP thread execution model for parallel construct

OpenMP Example: #pragma omp parallel for

```c
#pragma omp parallel for \
            schedule(static,1) \
            num_threads(3)
  for (int i = 0; i < 15; i++) {
    a[i] = 0.2 * a[i+1];
  }
```

Listing 2: OpenMP worksharing `for` construct

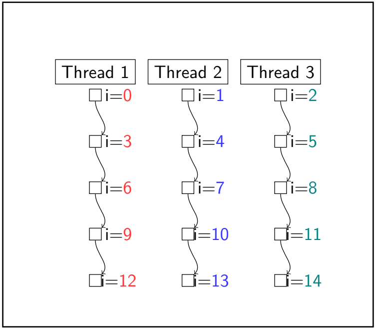

Figure 2: OpenMP thread execution model for `parallel for` construct

OpenMP Example: #pragma omp parallel for

```
#pragma omp parallel for \
            schedule(static, 4 ) \
            num_threads(3)
  for (int i = 0; i < 15; i++) {
    a[i] = 0.2 * a[i+1];
  }
```

Listing 3: OpenMP worksharing `for` construct

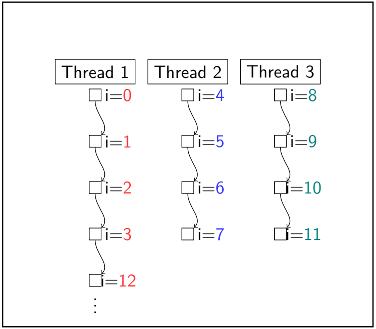

Figure 3: OpenMP thread execution model for `parallel for` construct with chunk size 4

```
1 #pragma omp parallel for \
2             schedule(static, 4 ) \
3             num_threads(3)
4   for (int i = 0; i < 15; i++) {
5       a[i]  = 0.2 * a[i+1] ;
6   }
```

Listing 3: OpenMP worksharing for construct

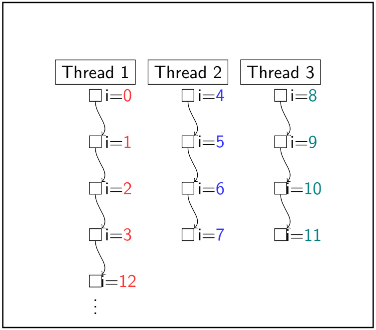

Figure 3: OpenMP thread execution model for parallel for construct with chunk size 4

Data race at (Line 5)!!

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 4: 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 5: DRB038: Example with Loop Carried Dependence

Common data 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];
}
```

Listing 6: 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 7: 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 8: 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

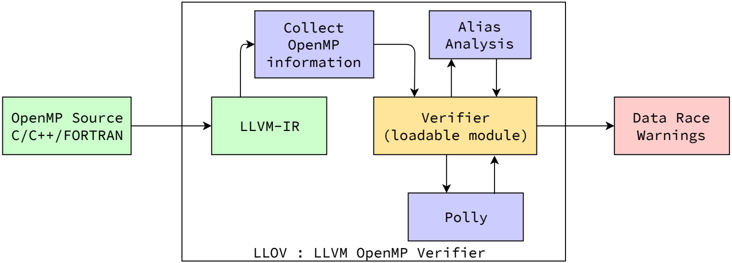

Figure 4: Flow Diagram of LLVM OpenMP Verifier (LLOV)

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 10: In-memory representation of a directive

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
```

```
<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 10: 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 : $\mathbf{S} = \{\mathtt{S0}(i,j) \rightarrow (i,j)\} \cap_{dom} \mathbf{I}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 13: Example with Loop Carried Dependence

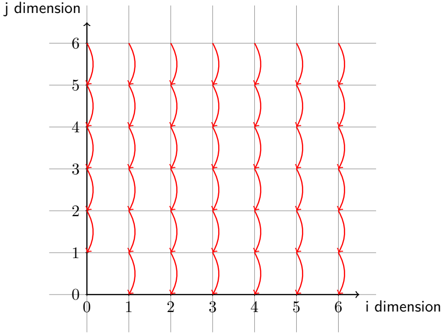

Figure 5: Dependence Polyhedra

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 14: Example with Loop Carried Dependence

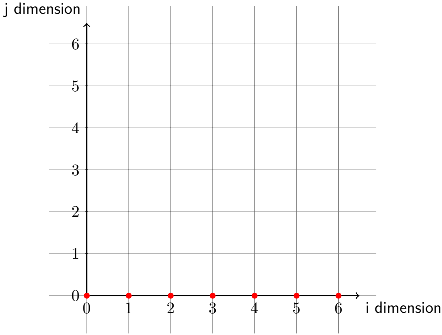

Figure 6: Projection of the Dependence Polyhedra on i-dimension

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

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 15: Example with Loop Carried Dependence

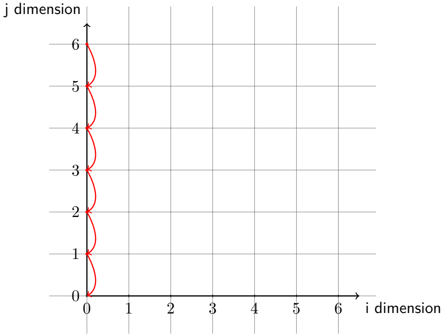

Figure 7: Projection of the Dependence Polyhedra on j-dimension

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

Related Work: OpenMP data race detection tools

Table 1: OpenMP Race Detection Tools: A Short Survey

Tools	Infrastructure	Analysis Type
HELGRIND [Valgrind-project, 2007b]	Valgrind	Dynamic
VALGRIND DRD [Valgrind-project, 2007a]	Valgrind	Dynamic
TSAN [Serebryany and Iskhodzhanov, 2009]	LLVM/GCC	Dynamic
SWORD [Atzeni et al., 2018]	LLVM	Dynamic
ROMP [Gu and Mellor-Crummey, 2018]	Dyninst	Dynamic
OMPVERIFY [Basupalli et al., 2011]	AlphaZ	Static
POLYOMP [Chatarasi et al., 2015]	PET	Static
DRACO [Ye et al., 2018]	ROSE	Static
ARCHER [Atzeni et al., 2016]	LLVM	Hybrid

Related Work: OpenMP data race detection tools

Table 1: OpenMP Race Detection Tools: A Short Survey

Tools	Infrastructure	Analysis Type
HELGRIND [Valgrind-project, 2007b]	Valgrind	Dynamic
VALGRIND DRD [Valgrind-project, 2007a]	Valgrind	Dynamic
TSAN [Serebryany and Iskhodzhanov, 2009]	LLVM/GCC	Dynamic
SWORD [Atzeni et al., 2018]	LLVM	Dynamic
ROMP [Gu and Mellor-Crummey, 2018]	Dyninst	Dynamic
OMPVERIFY [Basupalli et al., 2011]	AlphaZ	Static
POLYOMP [Chatarasi et al., 2015]	PET	Static
DRACO [Ye et al., 2018]	ROSE	Static
ARCHER [Atzeni et al., 2016]	LLVM	Hybrid

A static OpenMP data race checker was due 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
- Can detect races in all program paths

Advantage of Static tools over Dynamic tools

Static tools have the following advantages over dynamic tools:

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

Advantage of Static tools over Dynamic tools

Static tools have the following advantages over dynamic tools:

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

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 2: 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 3: 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 3: 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 4: 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 5: 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 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 6: 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

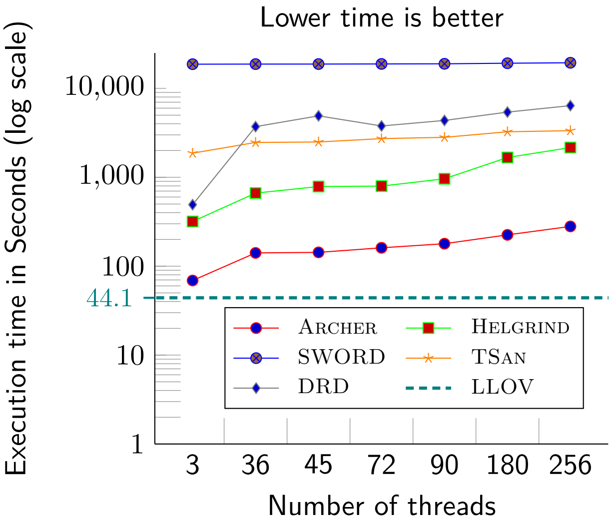

Figure 8: DataRaceBench v1.2 total execution time by different tools on logarithmic scale

Results: DataRaceBench v1.2 runtime

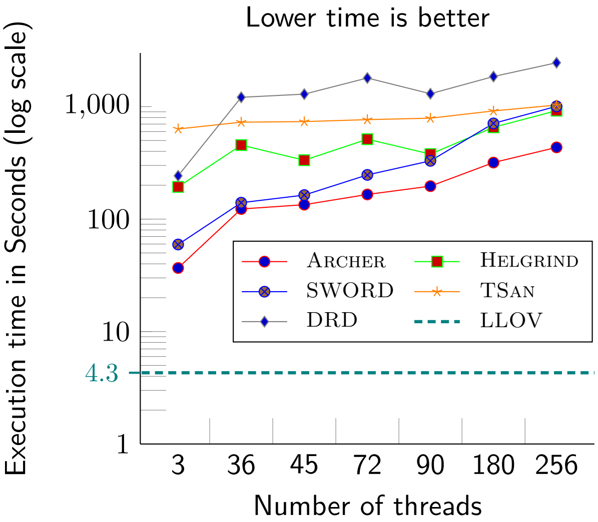

Figure 9: DataRaceBench v1.2 total time taken by different tools for common 61 kernels on logarithmic scale

Results: OmpSCR v2.0 comparison

Table 7: 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

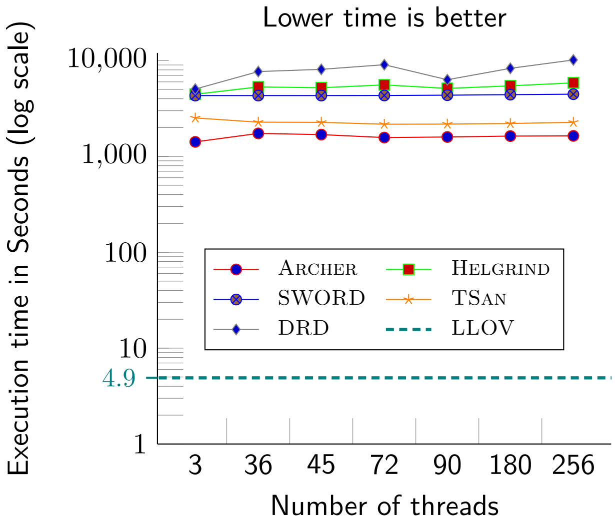

Figure 10: OmpSCR v2.0 total execution time by different tools on logarithmic scale

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]

Table 8: 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

Possible Extensions to LLOV

Working on

- Increase coverage- handle more OpenMP pragmas

Possible Extensions to LLOV

Working on

- Increase coverage- handle more OpenMP pragmas
  - *atomic*, *master*, *single*, and *critical* constructs

Possible Extensions to LLOV

Working on

- Increase coverage- handle more OpenMP pragmas
  - *atomic*, *master*, *single*, and *critical* constructs
  - Device offloading constructs (for GPUs)

Possible Extensions to LLOV

Working on

- Increase coverage- handle more OpenMP pragmas
  - *atomic*, *master*, *single*, and *critical* constructs
  - Device offloading constructs (for GPUs)
  - Tasking constructs

Possible Extensions to LLOV

Working on

- Increase coverage- handle more OpenMP pragmas
  - *atomic*, *master*, *single*, and *critical* constructs
  - Device offloading constructs (for GPUs)
  - Tasking constructs
- Use approximate dependece analysis of LLVM

Possible Extensions to LLOV

Working on

- Increase coverage- handle more OpenMP pragmas
  - *atomic*, *master*, *single*, and *critical* constructs
  - Device offloading constructs (for GPUs)
  - Tasking constructs
- Use approximate dependece analysis of LLVM
- Use May-Happen-in-Parallel analysis for data race detection

Possible Extensions to LLOV

Working on

- Increase coverage- handle more OpenMP pragmas
  - *atomic*, *master*, *single*, and *critical* constructs
  - Device offloading constructs (for GPUs)
  - Tasking constructs
- Use approximate dependece analysis of LLVM
- Use May-Happen-in-Parallel analysis for data race detection

Possible Extensions to LLOV

Working on

- Increase coverage- handle more OpenMP pragmas
  - *atomic*, *master*, *single*, and *critical* constructs
  - Device offloading constructs (for GPUs)
  - Tasking constructs
- Use approximate dependece analysis of LLVM
- Use May-Happen-in-Parallel analysis for data race detection

Challenges in using LoopAccessAnalysis of LLVM!

- Iteration distance is not available
- Dependence distance not computed for backward dependences
- How to construct direction vectors using LoopAccessAnalysis?

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/

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.

Basupalli, V., Yuki, T., Rajopadhye, S., Morvan, A., Derrien, S., Quinton, P., and Wonnacott, D. (2011).
ompverify: polyhedral analysis for the openmp programmer.
In *International Workshop on OpenMP*, pages 37–53, Berlin, Heidelberg. Springer, Springer Berlin Heidelberg.

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

References II

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.

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, pages 61:1–61:12, Piscataway, NJ, USA. IEEE Press.

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

References III

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.

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. 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 IV

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].

Ye, F., Schordan, M., Liao, C., Lin, P.-H., Karlin, I., and Sarkar, V. (2018).
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. IEEE, IEEE.

Extra Slides!

Table 9: Comparison of OpenMP pragma handling by OpenMP aware tools. (Y for Yes, N for No)

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

Data race detection literature

Table 10: 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 OpenMP's Data Mapping Constructs	2019
SC	OMPRacer: A Scalable and Precise Static Race Detector for OpenMP Programs	2020

Vectorization vs Parallelization

Table 11: Dependence Types

Dependence Type	RAW	WAR

Laxically Backward	`for(int i=0;i<N;i++){`
	`  ... = A[i];`
	`  A[i+4] = ...;`
	`}`
		`for(int i=0;i<N;i++){`
		`  A[i] = ...;`
		`  ... = A[i+4];`
		`}`

Laxically Forward	`for(int i=0;i<N;i++){`
	`  A[i+4] = ...;`
	`  ... = A[i];`
	`}`
		`for(int i=0;i<N;i++){`
		`  ... = A[i];`
		`  A[i-4] = ...;`
		`}`

Table 12: Vectorization and Parallelization legality

Dependence Type	Vectorization Legality	Parallelization Legality
Laxically Backward	Legal for VF $<$ Iteration Distance	Restrictive schedule with PF $<$ Iteration Distance
Laxically Forward	Legal with VF $= \infty$	Restrictive schedule with PF $<$ Iteration Distance

Vectorization Example

```
for (int i = 0; i < 20; i++) {
  for (int j = 4; j < 20; j++) {
    b[i][j]=b[i][j-4] + x;
  }
}
```

Listing 16: Example for Vectorization

Vectorization Example

```c
for (int i = 0; i < 20; i++) {
  for (int j = 4; j < 20; j++) {
    b[i][j]=b[i][j-4] + x;
  }
}
```

Listing 16: Example for Vectorization

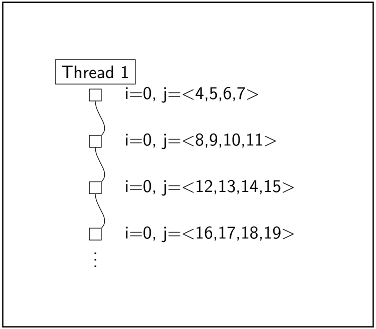

Figure 11: SIMD execution model within CPU

Parallelization Example:

```
for (int i = 0; i < 20; i++) {
#pragma omp parallel for \
            schedule(static,4) \
            num_threads(3)
  for (int j = 4; j < 20; j++) {
      b[i][j] = b[i][j-4] + x;
  }
}
```

Listing 17: OpenMP worksharing `for` construct (incorrect parallelization)

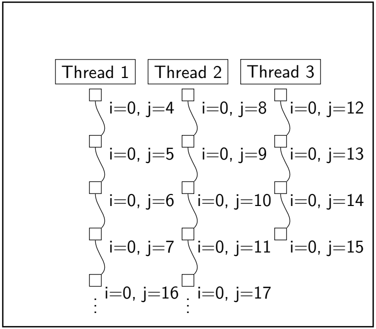

Figure 12: OpenMP thread execution schedule for `parallel for` construct with data races

Parallelization Example:

```c
for (int i = 0; i < 20; i++) {
#pragma omp parallel for \
            schedule(static,4) \
            num_threads(3)
  for (int j = 4; j < 20; j++) {
      b[i][j] = b[i][j-4] + x;
#pragma omp barrier
  }
}
```

Listing 18: OpenMP worksharing `for` construct with barrier (incorrect parallelization)

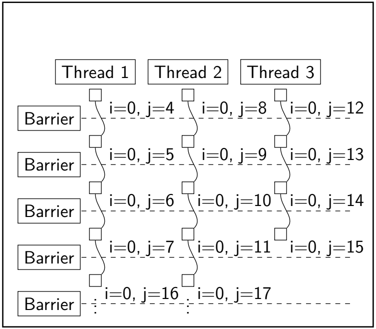

Figure 13: Possible OpenMP thread execution schedule for `parallel for` construct with chunking

Parallelization Example:

```
for (int i = 0; i < 20; i++) {
#pragma omp parallel for \
            schedule(static,4) \
            num_threads(3)
  for (int j = 4; j < 20; j++) {
      b[i][j] = b[i][j-4] + x;
#pragma omp barrier
  }
}
```

Listing 18: OpenMP worksharing for construct with barrier (incorrect parallelization)

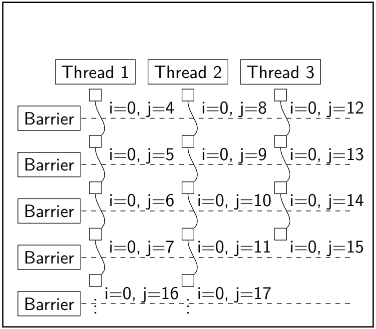

Figure 13: Possible OpenMP thread execution schedule for parallel for construct with chunking

Incorrect parallelization! Barrier can not be placed inside a parallel for construct.

Correct Parallelization Example:

```
#pragma omp parallel for \
            schedule(static,4) \
            num_threads(3)
for (int i = 0; i < 20; i++) {
  for (int j = 4; j < 20; j++) {
     b[i][j] = b[i][j-4] + x;
  }
}
```

Listing 19: OpenMP worksharing `for` construct

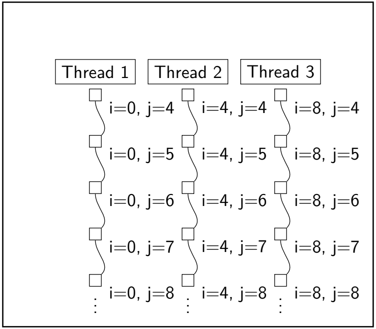

Figure 14: OpenMP thread execution schedule for `parallel for` construct with chunking

Correct Parallelization and Vectorization Example:

```
#pragma omp parallel for \
            schedule(static,4) \
            num_threads(3)
for (int i = 0; i < 20; i++) {
#pragma omp simd simdlen(4)
  for (int j = 4; j < 20; j++) {
      b[i][j] = b[i][j-4] + x;
  }
}
```

Listing 20: OpenMP worksharing `for` construct

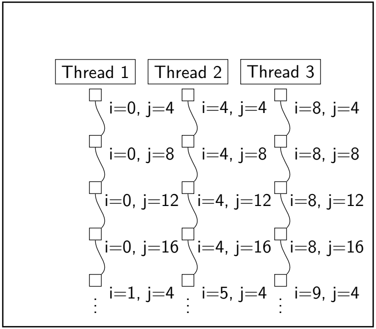

Figure 15: OpenMP thread execution schedule for `parallel for` construct with chunking

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.