On-the-fly Data Race Detection with the Enhanced OpenMP Series-Parallel Graph

Nader Boushehrinejad, Adarsh Yoga, Santosh Nagarakatte

Rutgers University

IWOMP 20

What is a Data Race?

A data race occurs when two memory accesses:

- Access the same location
- Both accesses are in parallel
- At least one is a write (conflicting access)

The execution of a program contains a *data race* if it contains two conflicting actions in different threads, at least one of which is not atomic, and neither happens before the other. Any such data race results in undefined behavior.

C17 Standard

What Problems are Caused by Data Races?

- Undefined behavior

The execution of a program contains a *data race* if it contains two conflicting actions in different threads, at least one of which is not atomic, and neither happens before the other. Any such data race results in undefined behavior.

C17 Standard

What Problems are Caused by Data Races?

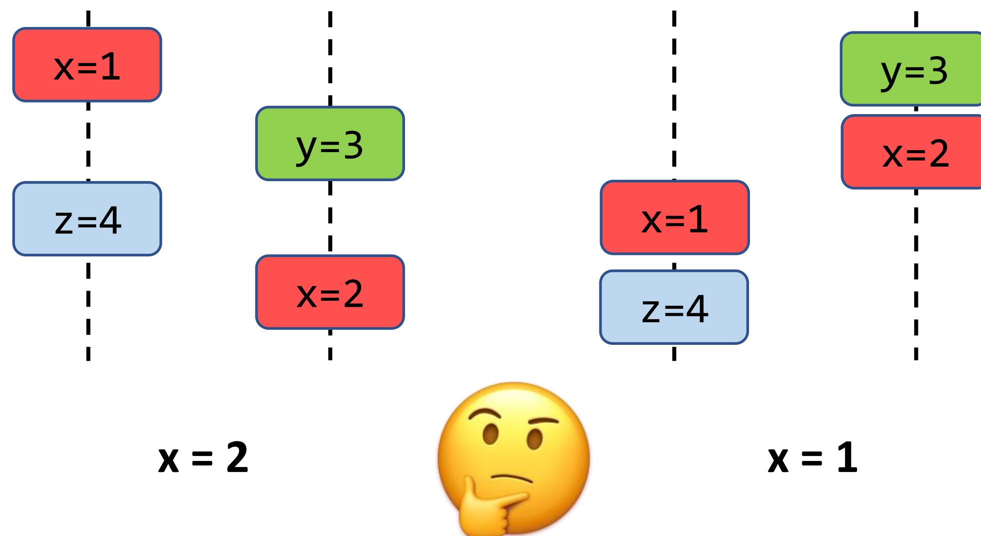
- Undefined behavior
- Execution dependent on memory model

NOTE 18 It can be shown that programs that correctly use simple mutexes and `memory_order_seq_cst` operations to prevent all data races, and use no other synchronization operations, behave as though the operations executed by their constituent threads were simply interleaved, with each value computation of an object being the last value stored in that interleaving. This is normally referred to as "sequential consistency". However, this applies only to data-race-free programs, and data-race-free programs cannot observe most program transformations that do not change single-threaded program semantics. In fact, most single-threaded program transformations continue to be allowed, since any program that behaves differently as a result must contain undefined behavior.

C17 Standard

What Problems are Caused by Data Races?

- Undefined behavior
- Execution dependent on memory model
- Non-determinism




```
Thread A
{
  x=1;
  z=4;
}
```

```
Thread B
{
  y=3;
  x=2;
}
```

Identifying Data Races is Challenging

- Exponential number of interleavings
  - Approximately $t^{t \times n}$ for t threads with n instructions

Exactly locating the feasible general races or data races is an NP-hard problem. This result implies that the apparent races, which are simpler to locate, must be detected for debugging in practice.

Netzer & Miller, 92

Contributions

- Detect apparent races in OpenMP applications
  - For a given input, run once, identify races in other interleavings
  - Devise different access history management strategies
- Proposes a novel data structure (EOSPG)
  - Encodes logical series-parallel relations
  - Supports structured and unstructured OpenMP constructs
- Open-source:
  - https://github.com/rutgers-apl/omp-racer

On-the-fly Data Race Detection Overview

May execute in Parallel	Access history
L3 < L6	(Wr, L3)
L3 < L10	(Wr, L6)
L6 || L10	(Rd, L10)
	(Wr, L10)

```
1  #pragma omp single
2  {
3    x=0;
4    #pragma omp task
5    {
6      x=1;
7    }
8    #pragma omp task
9    {
10     x+=1;
11     print(x);//x=1 or x=2
12   }
13 }
```

Data race

- Must implement the following mechanisms:
  - Check if two memory accesses execute in parallel
  - Keep track of previous memory accesses

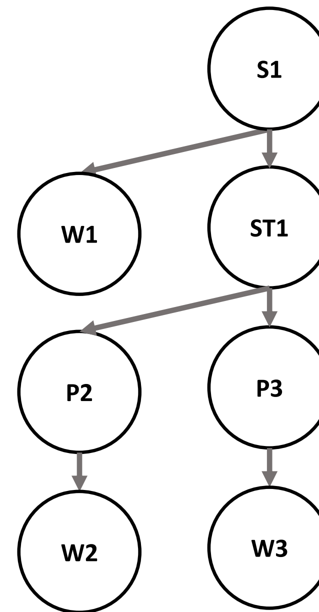
Capturing Logical Series-Parallel Relations

Enhanced OpenMP Series-Parallel Graph (EOSPG)

- To detect apparent races requires a data structure that:
  - Encodes logical series-parallel relations
  - Logical relations are independent of thread interleaving
- The EOSPG captures these relations:
  - Encodes the semantics of different OpenMP constructs
  - Some constructs have interesting semantics

EOSPG Overview

- Encode series-parallel relations between program fragments
  - Each fragment is represented by a W-node
  - W-nodes are always leaf nodes
  - Internal nodes (S, P, ST) and edges encode relations between W-nodes
  - Can query the logical series-parallel of any pair of W-nodes in O(h)



W1: lines 2–3; W2: lines 5–7; W3: lines 9–12

```
1   #pragma omp single
2   {
3     x=0
4     #pragma omp task
5     {
6       x=1;
7     }
8     #pragma omp task
9     {
10      x+=1;
11      print(x);//x=1 or x=2
12    }
13  }
```

A fragment is the longest sequence of instructions in the dynamic execution before encountering an OpenMP construct

Example: On-the-fly EOSPG Construction

```
1 int main(){
2   int a[4];
3   int psum[2];
4   int sum;
5   #pragma omp parallel num_threads(2)
6   {
7     #pragma omp for schedule(dynamic, 1)
8       for (int i=0; i < 4; ++i)
9       {
10        a[i] = i;
11      }//implicit barrier
12    #pragma omp single nowait
13    {
14      #pragma omp task
15      {
16        #pragma omp task
17        {
18          psum[1] = a[2] + a[3];
19        }
20        psum[0] = a[0] + a[1];
21      }
22      #pragma omp taskwait
23      sum = psum[1] + psum[0];
24    }
25  }//barrier
26  printf("sum = %d\n", sum);
27  return 0;
28 }
```

Read-Write Data race on psum[1] lines 18-23

S-nodes encode serial relations:

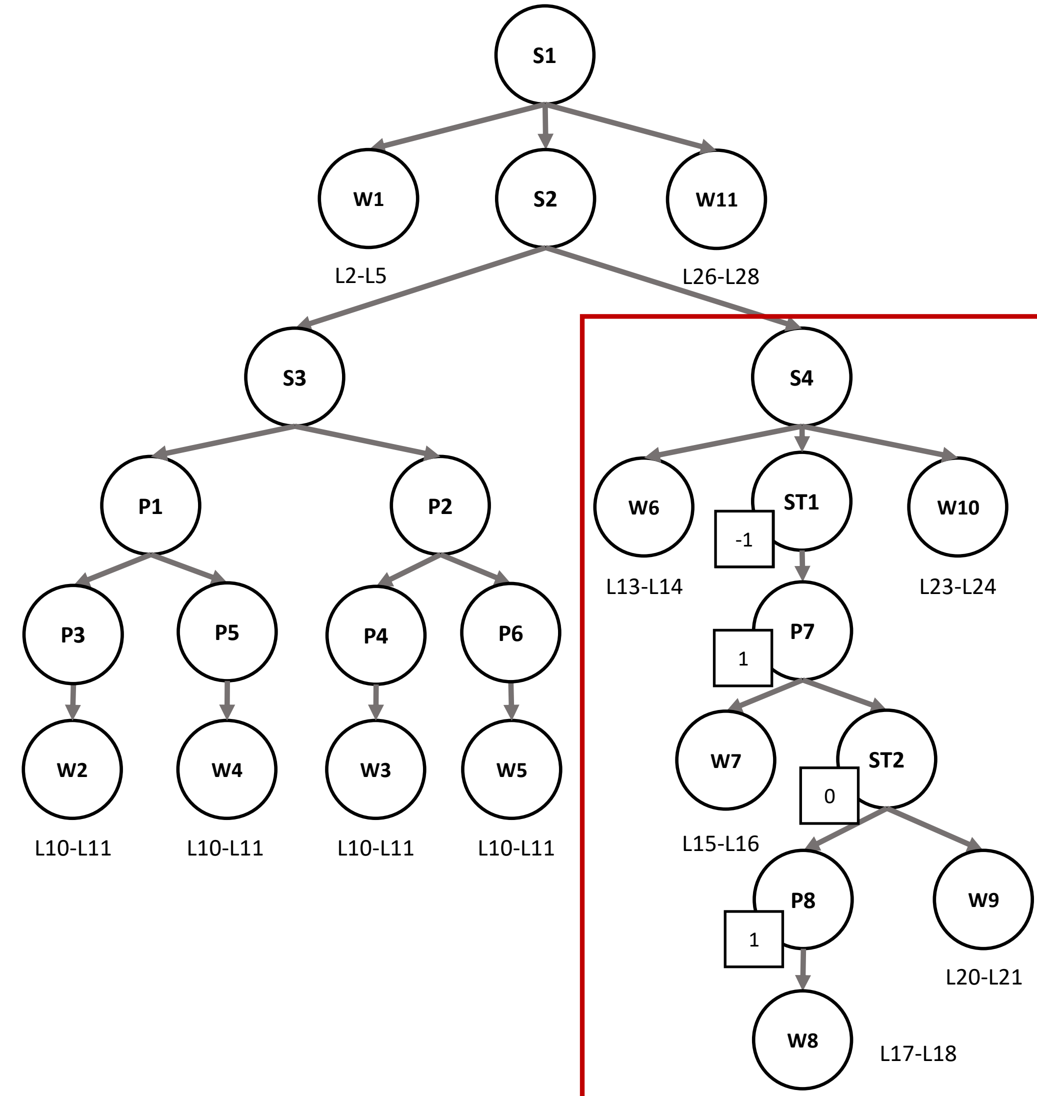
- Between subtree and right siblings and their descendants
- Parallel, Barrier

P-nodes encode parallel relations:

- Between subtree and right siblings and their descendants
- Parallel, loops, task

ST-nodes partition descendant W-nodes into series and parallel

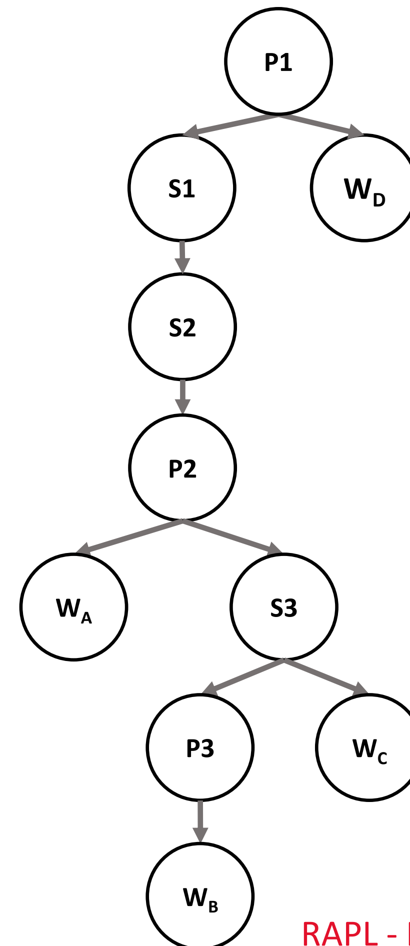
- Between subtree and right siblings and their descendants
- Task, taskwait



Modeling Taskgroup With the EOSPG

- Synchronizes current task with its children tasks and all their descendant tasks

```
1  #pragma omp single nowait
2  {
3     #pragma omp taskgroup
4     {
5       #pragma omp task
6       {
7         A();
8         #pragma omp task
9         {
10          B();
11        }
12        C();
13      }
14    }
15    D();
16  }
17 }
```



Program order:
A < C
A < B

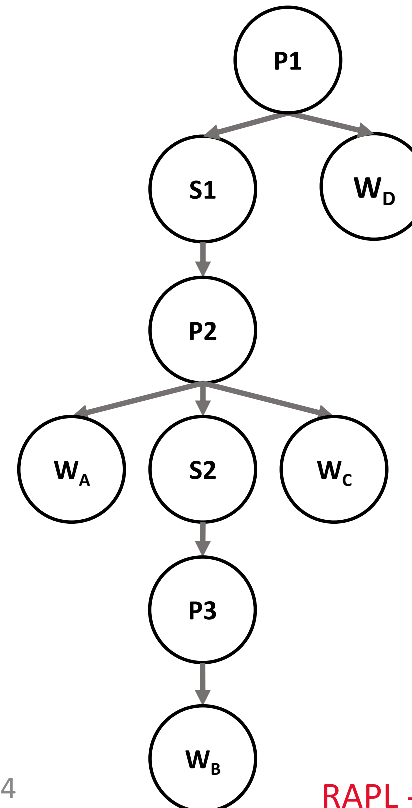
Taskgroup
A < D
B < D
C < D

S-node S1 captures the semantics of the taskgroup at line 3

Modeling Taskwait With the EOSPG

- Synchronizes the current task with its child tasks
  - Does not synchronize current task with its descendant tasks

```
1 #pragma omp single nowait
2 {
3    #pragma omp task
4    {
5      A();
6      #pragma omp task
7      {
8        B();
9       }
10     #pragma omp taskwait
11     C();
12   }
13   #pragma omp taskwait
14   D();
15  }
16 }
```



Program order:
A < C
A < B

L10 taskwait:
B < C

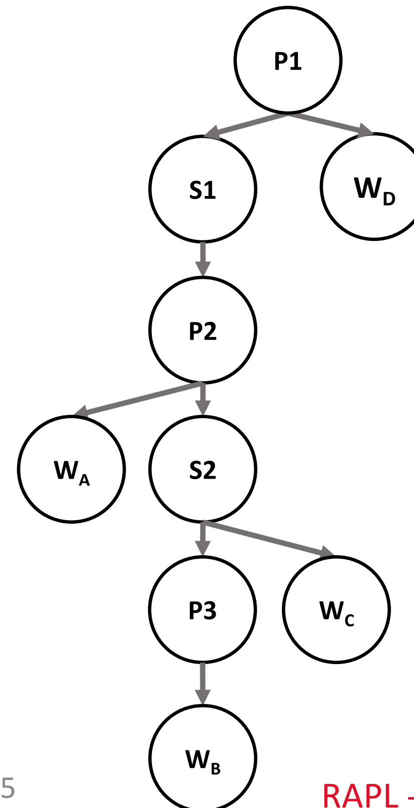
L13 taskwait:
A < D
C < D

A < D
C < D
B < D

Modeling Taskwait With the EOSPG

- Synchronizes the current task with its child tasks
  - S-nodes are not sufficient to model all taskwaits

```
1 #pragma omp single nowait
2 {
3    #pragma omp task
4    {
5      A();
6      #pragma omp task
7      {
8        B();
9       }
10     //#pragma omp taskwait
11     C();
12   }
13   #pragma omp taskwait
14   D();
15  }
16 }
```



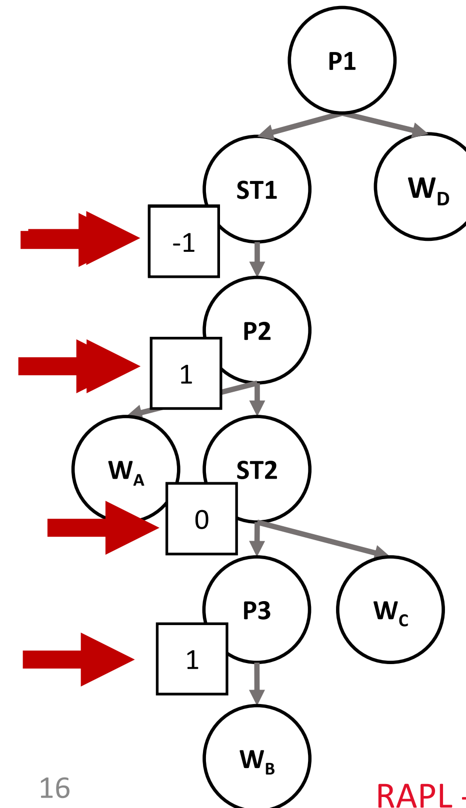
L13 taskwait:
A < D
C < D

A < D
C < D
B || D

Modeling Taskwaits Using the EOSPG

- Encode taskwait encounter in the ST-node
- Infer series-parallel relation from st-val on path to ST-node

```
1 #pragma omp single nowait
2 {
3    #pragma omp task
4    {
5      A();
6      #pragma omp task
7      {
8        B();
9       }
10     //#pragma omp taskwait
11     C();
12   }
13   #pragma omp taskwait
14   D();
15  }
16 }
```



Series-parallel relation is determined by computing sum of node st-vals on the path from LCA to left W-node

Positive → Parallel

0 or negative → Series

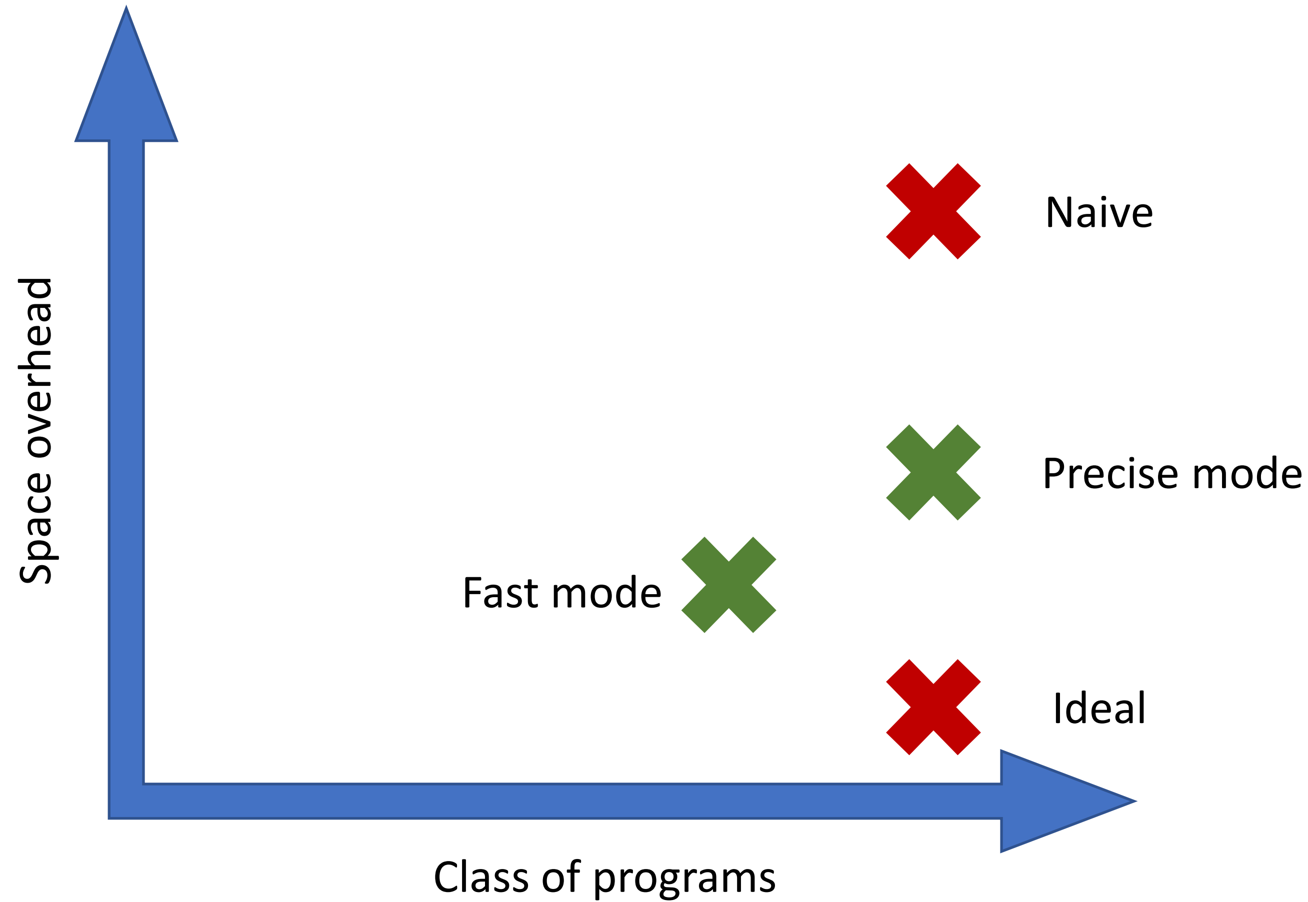
$PathSum(W_A, W_D) = 0 \rightarrow W_A < W_D$

$PathSum(W_C, W_D) = 0 \rightarrow W_C < W_D$

$PathSum(W_B, W_D) = 1 \rightarrow W_B \,||\, W_D$

Access History Management

Access History Management



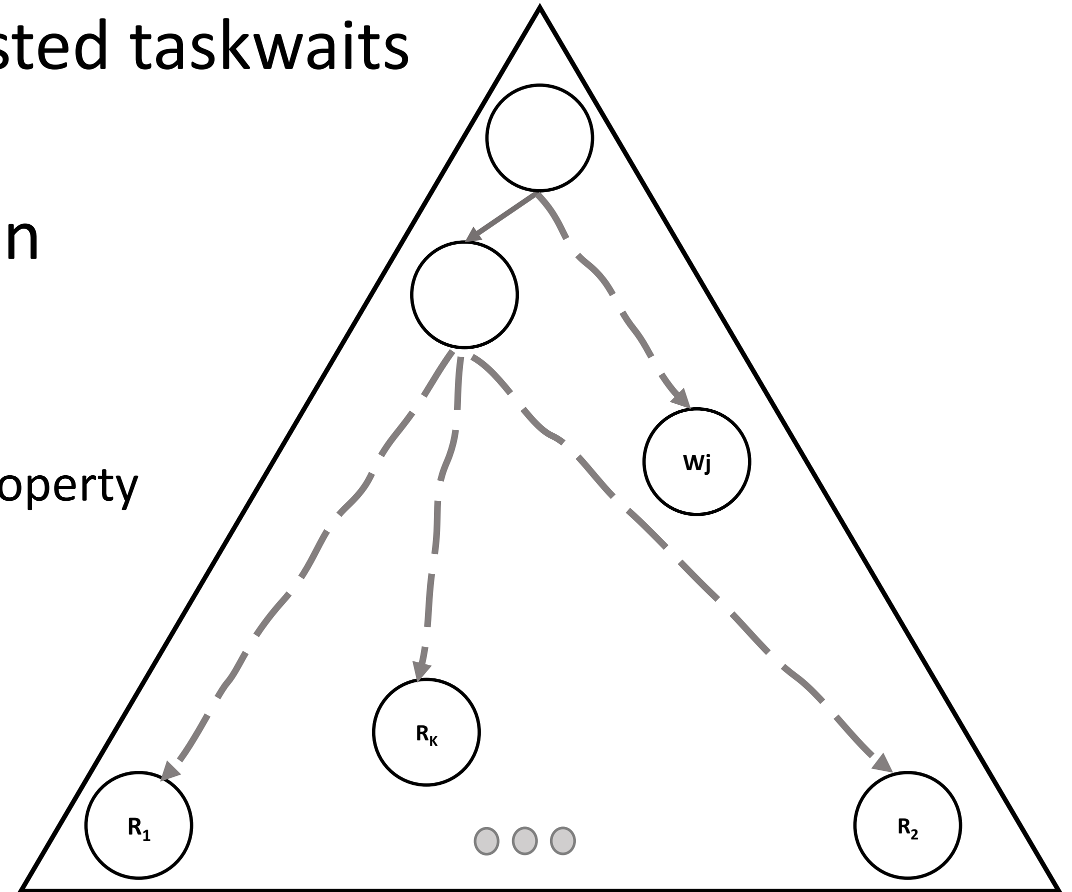
Access History Management in Fast Mode

- First, check if program has perfectly nested taskwaits
  - Treat all ST-nodes as S-nodes
- Constant metadata per memory location
  - Maintain the latest write
  - Maintain up to 2 previous parallel reads

May Execute in Parallel relation has transitive property

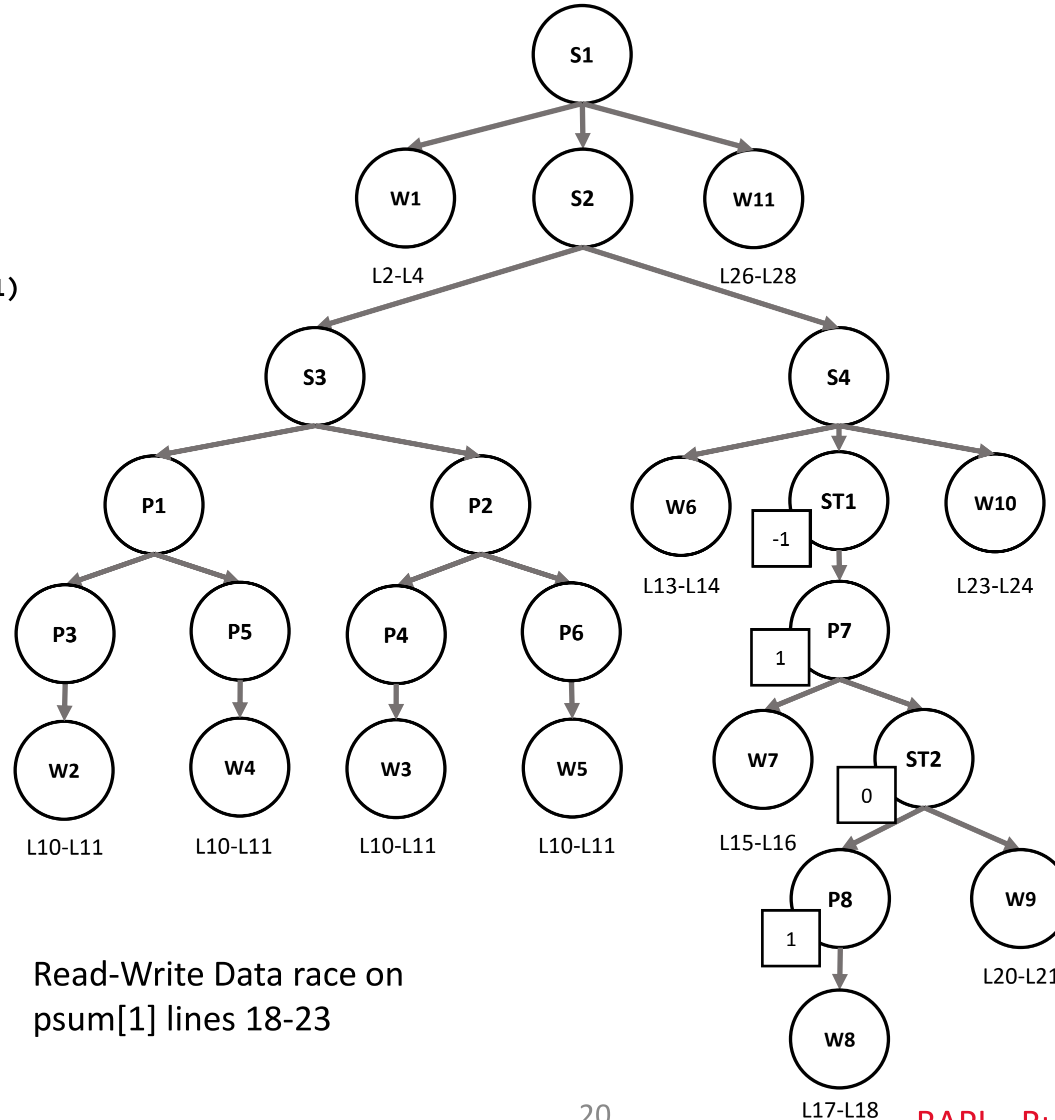
$R_1 || R_2 || R_K$

$W_i || R_1 \rightarrow R_1 || W_i$
or
$W_i || R_2 \rightarrow R_2 || W_i$



Example: Access History Management

```
1 int main(){
2   int a[4];
3   int psum[2];
4   int sum;
5   #pragma omp parallel num_threads(2)
6   {
7     #pragma omp for schedule(dynamic, 1)
8       for (int i=0; i < 4; ++i)
9       {
10        a[i] = i;
11      }//implicit barrier
12    #pragma omp single nowait
13    {
14      #pragma omp task
15      {
16        #pragma omp task
17        {
18          psum[1] = a[2] + a[3];
19        }
20        psum[0] = a[0] + a[1];
21      }
22      #pragma omp taskwait
23      sum = psum[1] + psum[0];
24    }
25  } //barrier
26  printf("sum = %d\n", sum);
27  return 0;
28 }
```



Read-Write Data race on psum[1] lines 18-23

Psum[1]
((Wr, W8),-,-)

Example: Access History Management

```
1 int main(){
2   int a[4];
3   int psum[2];
4   int sum;
5   #pragma omp parallel num_threads(2)
6   {
7     #pragma omp for schedule(dynamic, 1)
8       for (int i=0; i < 4; ++i)
9       {
10        a[i] = i;
11      }//implicit barrier
12    #pragma omp single nowait
13    {
14      #pragma omp task
15      {
16        #pragma omp task
17        {
18          psum[1] = a[2] + a[3];
19        }
20        psum[0] = a[0] + a[1];
21      }
22      #pragma omp taskwait
23      sum = psum[1] + psum[0];
24    }
25  } //barrier
26  printf("sum = %d\n", sum);
27  return 0;
28 }
```

S1
W1 S2 W11
L2-L4 L26-L28
S3 S4
P1 P2 W6 ST1 W10
-1
L13-L14 L23-L24
P3 P5 P4 P6 P7
1
W2 W4 W3 W5 W7 ST2
0
L10-L11 L10-L11 L10-L11 L10-L11 L15-L16
P8 W9
1
L20-L21
W8
L17-L18

Read-Write Data race on psum[1] lines 18-23

Psum[1]

((Wr, W8), (Rd, W10),-)

W8 || W10 → Report data race

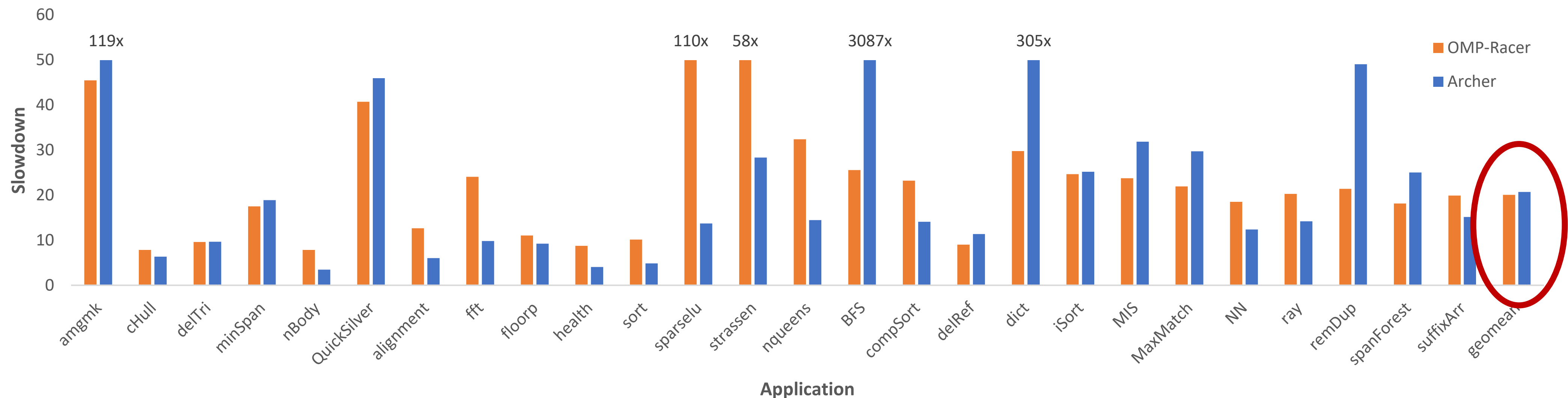
Evaluation

Evaluation

- How effective is OMP-Racer in detecting data races?
  - DataRaceBench 1.2.0 microbenchmarks (106/116)
  - Detect all Data Race with no false positives
  - Requires a single run to detect data races
- What are the performance overheads of OMP-Racer?
  - 26 OpenMP applications and benchmarks

    Coral, BOTS, and PBBS

Performance Overhead Comparison



- Overall, similar overhead to Archer in fast mode
- Without potentially requiring multiple executions

Related Work

- Lockset Algorithm [Cheng:SPAA98]
  - Detect data races in multithreaded programs with locks
- SPD3 and PTRacer [Raman:RV10,Yoga:FSE16]
  - DPST series-parallel graph
  - Supports structured task-based parallelism
- ROMP [Gu:SC18]
  - Enhances offset-span labeling for OpenMP data race detection
- Archer [Atzeni:IPDPS16]
  - State-of-the-art data race detector for OpenMP

Conclusion

- The EOSPG encodes logical series-parallel relations
  - Models OpenMP applications with unstructured parallelism
  - Can be used to design dynamic analysis tools for OpenMP applications
- OMP-Racer uses the EOSPG to detect apparent races
  - For a given input, run program once, identify in all interleavings
  - Different strategies for access history management
- Open-source: https://github.com/rutgers-apl/omp-racer