A Case for Parallelism Profilers and Advisers with What-If Analyses

Santosh Nagarakatte

Rutgers University, USA

@ Workshop on Dependable and Secure Software Systems @ ETH Zurich, October 2019

RUTGERS

Is Parallel Programming Hard, And, If So, What Can You Do About It?

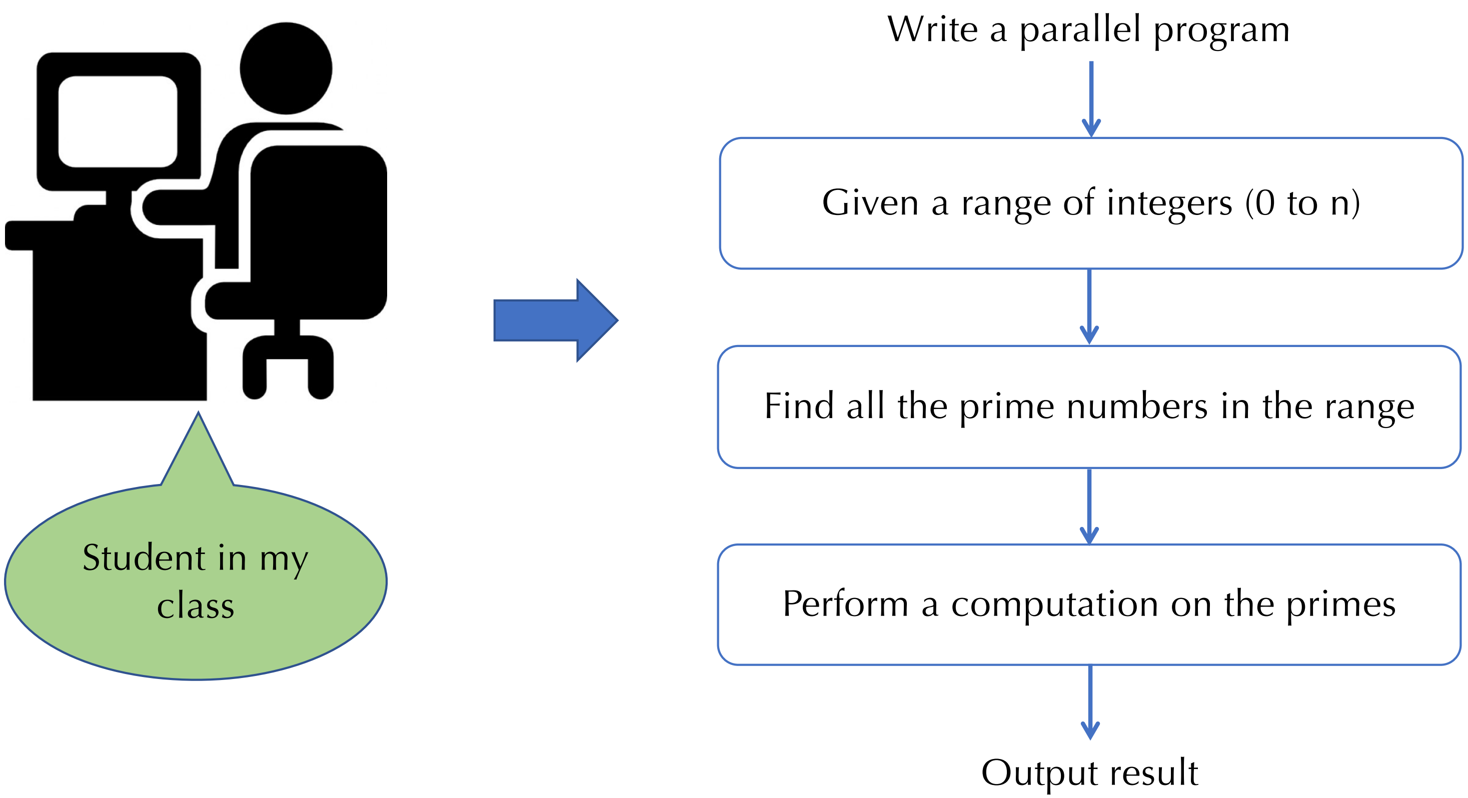
"Parallel programming has earned a reputation as one of the most difficult areas a hacker can tackle. Papers and textbooks warn of the perils of deadlock, livelock, race conditions, non-determinism, Amdahl's-Law limits to scaling, and excessive realtime latencies. And these perils are quite real; we authors have accumulated uncounted years of experience dealing with them, and all of the emotional scars, grey hairs, and hair loss that go with such experiences."

[McKenny:arXiv17]

Main reasons: use of the wrong abstraction, lack of performance analysis and debugging tools

RUTGERS

Illustrative Example



Illustrative Example – Writing a Parallel Program

OpenMP™

Feature rich

Work-Sharing | Tasking | SIMD | Offload

Incremental parallelization

```
#pragma omp parallel for
  for(int i=0; i<n; ++i) compute(i);
```

Illustrative Example

Student in my class

1	2	3	4	……..	n

Identify the number of processors on the machine (4)

Divide the range into 4 parts and perform computation

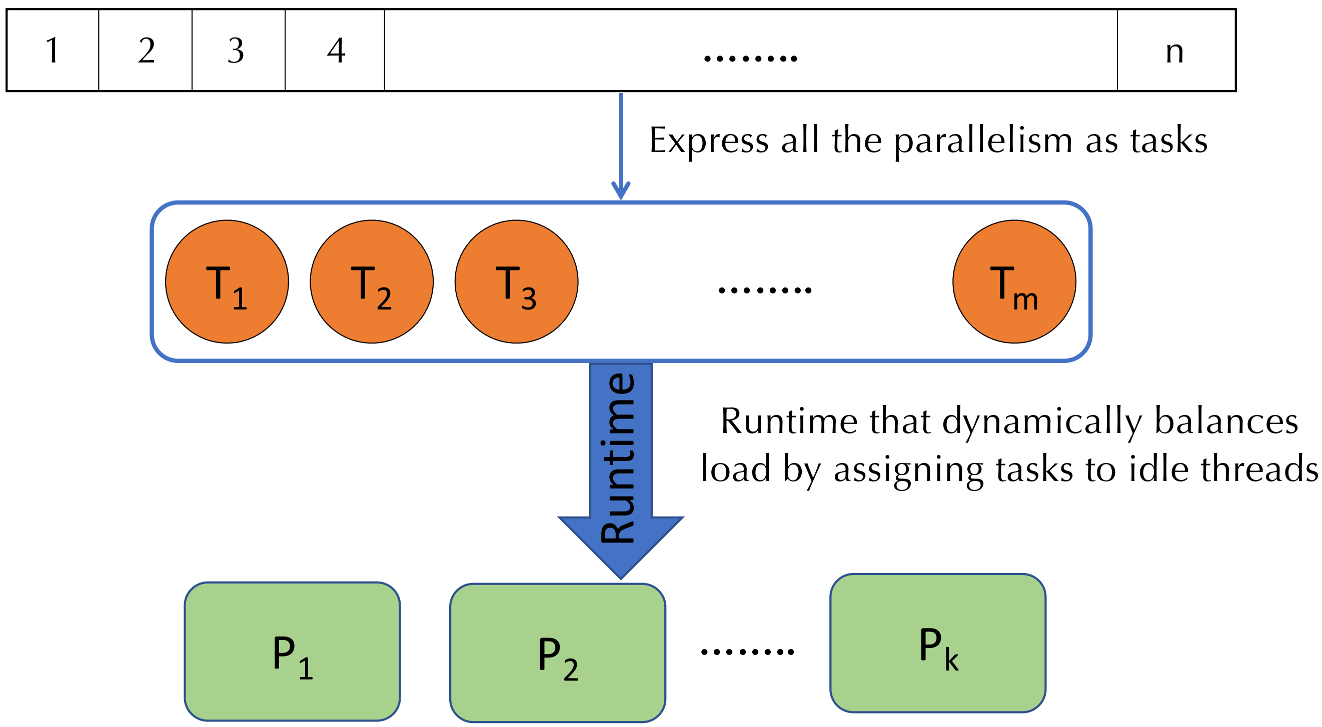
Run: ./primes

Speedup: 1.8X over serial execution

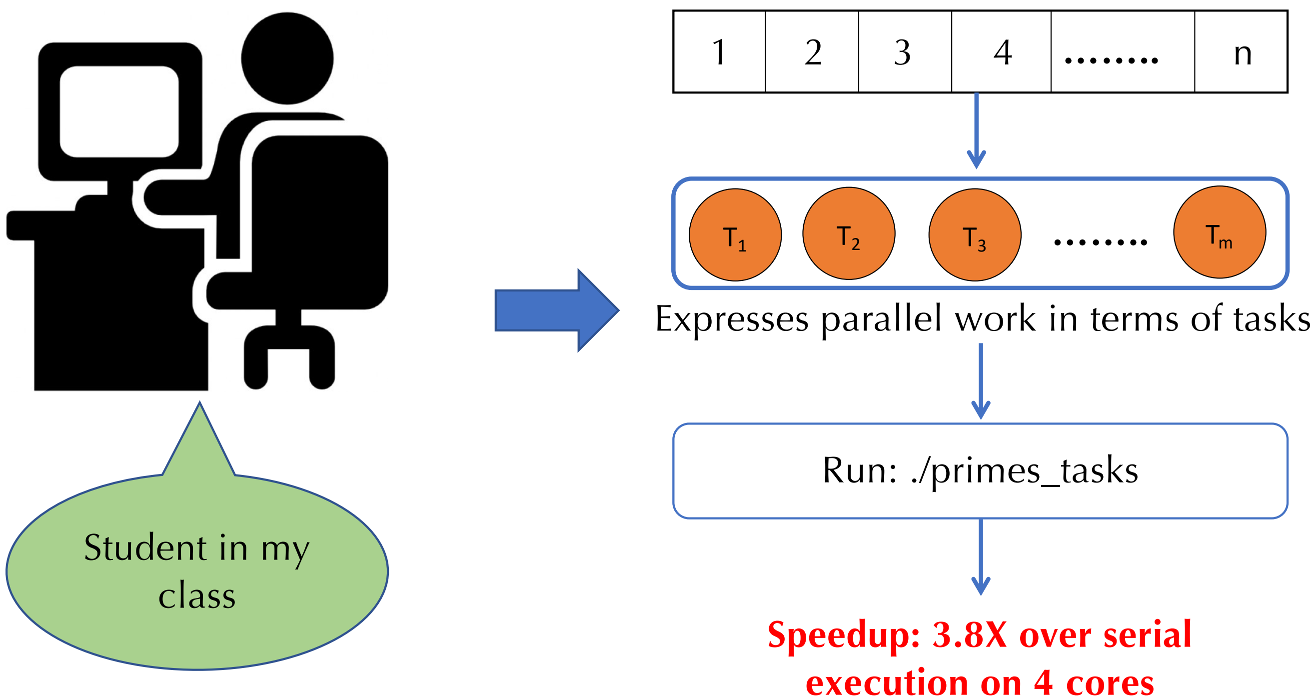
Why?

Load Imbalance

Need to write Performance Portable Code - Advocacy for Task Parallelism



Illustrative Example



Is it performance portable?

Performance Debugging Tools

Coz GProf OProfile ARMMap

arm DDT

PARALLEL STUDIO XE

PARALLELWARE TRAINER

Score-P

scalasca

VAMPIR

- Most of them provide info on frequently executed regions.
- Critical path information is useful
- Coz [SOSP 2015]: Identifies if a line of code matters in increasing speedup on a given machine.

Advisor

RUTGERS

Our Parallelism Profilers and Advisers: TaskProf & OMP-WHIP [FSE 2017, SC 2018, PLDI 2019]

- Making a case for measuring logical parallelism

Series-parallel relations + fine-grained measurements is a performance model

- Where should programmer focus?

Regions with low parallelism => serialization. Critical path!

Profiler

- Does it matter?

Automatically identify regions to increase parallelism to a threshold

What-if Analyses - mimic the effect of parallelization

Adviser

Differential analyses to identify regions with secondary effects

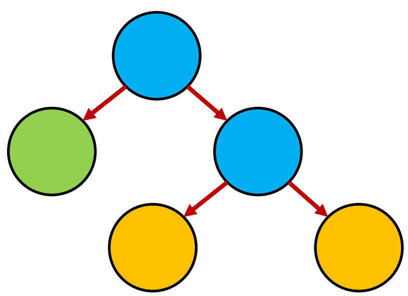
RUTGERS

General for multiple parallelism models. This talk focuses on OpenMP

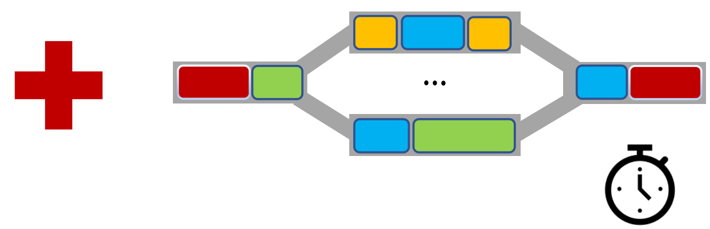
Performance Model for Logical Parallelism and What-If Analyses

Performance Model for Computing Parallelism

- Profile on a machine with low core count and identify scalability bottlenecks
- OSPG: Logical series-parallel relations between parts of a OpenMP program
  - Inspired by prior work: DPST [PLDI 2012], SP Parse tree [SPAA 2015]



OSPG



Fine-grained measurements

OpenMP Series Parallel Graph (OSPG)

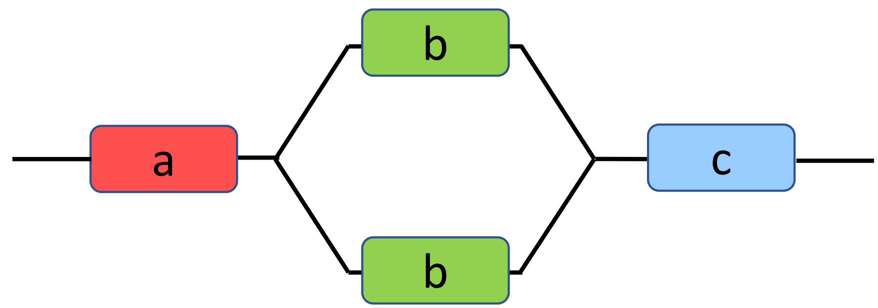
- A data structure to capture series-parallel relations
  - Inspired by Dynamic Program Structure Tree [PLDI 2012]
  - OSPG is an ordered tree in the absence of task dependencies in OpenMP
- Handles the combination of work-sharing (fork-join programs with threads) and tasking
- Precisely captures the semantics of OpenMP
  - Three kinds of nodes : W, S, and P nodes similar to Async, Finish, and Step nodes in the DPST

Code Fragments in OpenMP Programs

OpenMP code snippet

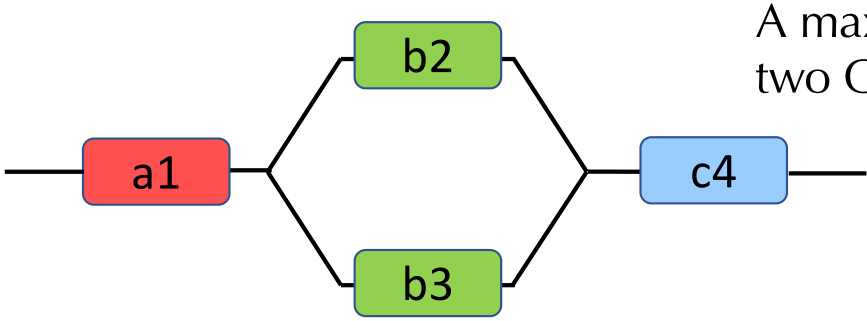
```
…
a();
#pragma omp parallel
  b();
c();
…
```

Execution structure



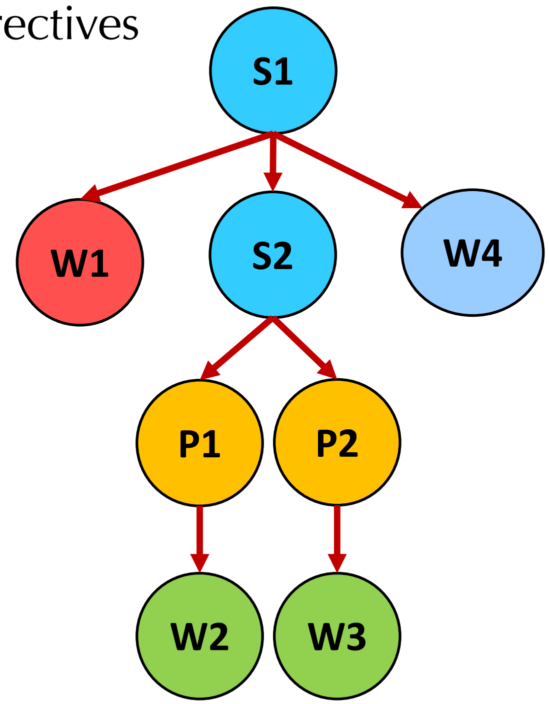
A code fragment is the longest sequence of instructions in the dynamic execution before encountering an OpenMP construct

Capturing Series-Parallel Relation with the OSPG



W-nodes capture computation

A maximal sequence of dynamic instructions between two OpenMP directives



P-nodes capture the parallel relation

Nodes in the sub-tree of a P-node logically executes in **parallel** with right siblings of the P-node

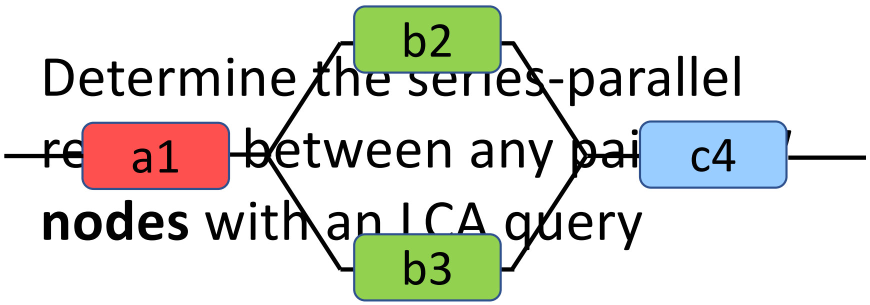
S-nodes capture the series relation

Nodes in the sub-tree of a S-node logically executes in **series** with right siblings of the S-node

RUTGERS

Capturing Series-Parallel Relation with the OSPG

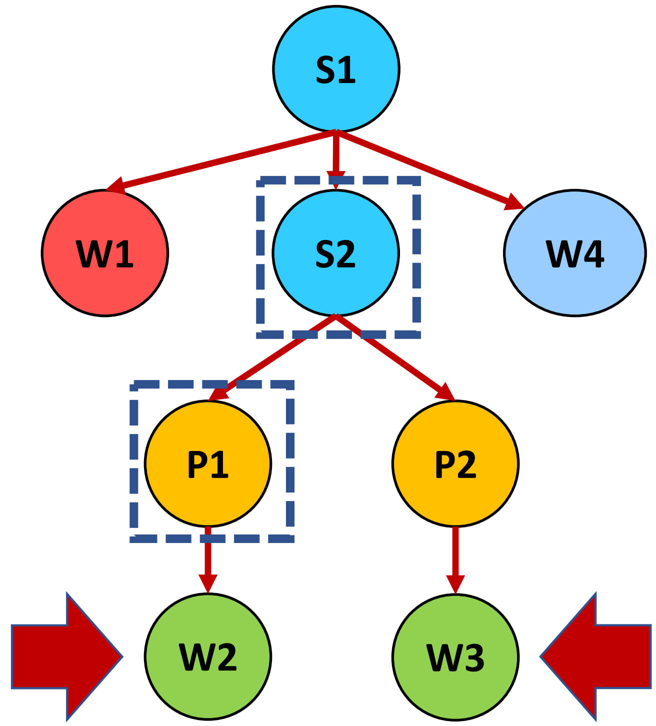
Determine the series-parallel re between any pai **nodes** with an LCA query



Check the type of the LCA's child on the path to the left w-node. If it's a **p-node**, they execute in **parallel**. Otherwise, they execute in **series**

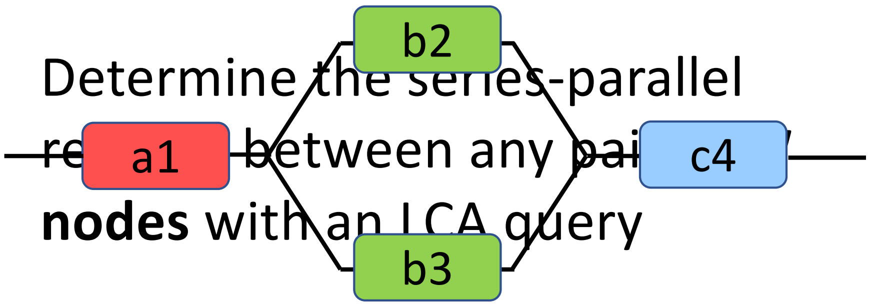
S2 = LCA(W2,W3)

P1 = Left-Child(S2,W2,W3)



Capturing Series-Parallel Relation with the OSPG

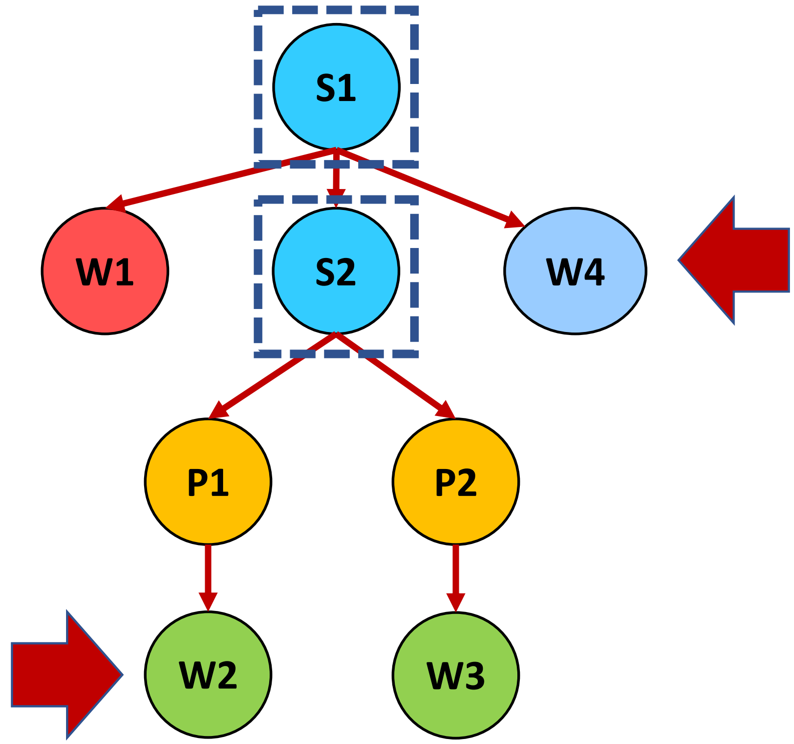
Determine the series-parallel re between any pai **nodes** with an LCA query



Check the type of the LCA's child on the path to the left w-node. If it's a **p-node**, they execute in **parallel**. Otherwise, they execute in **series**

S1 = LCA(W2,W4)

S2 = Left-Child(S1,W2,W4)



RUTGERS

Profiling an OpenMP Merge Sort Program

- Merge sort program parallelized with OpenMP

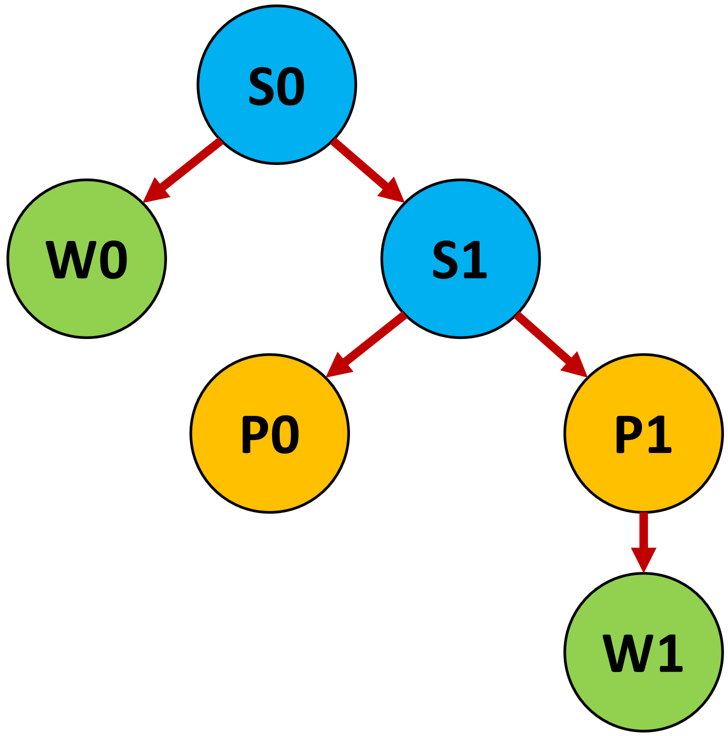
```
void main(){
  int* arr = init(&n);
#pragma omp parallel
  #pragma omp single
    mergeSort(arr, 0, n);
}
```

```
void mergeSort(int* arr, int s, int e){
  if (n <= CUT_OFF)
     serialSort(arr, s, e);
  int mid = s + (e-s)/2;
  #pragma omp task
     mergeSort(arr, s, mid);
  #pragma omp task
     mergeSort(arr, mid+1, e);
  #pragma omp taskwait
  merge(arr, s, e);
}
```

RUTGERS

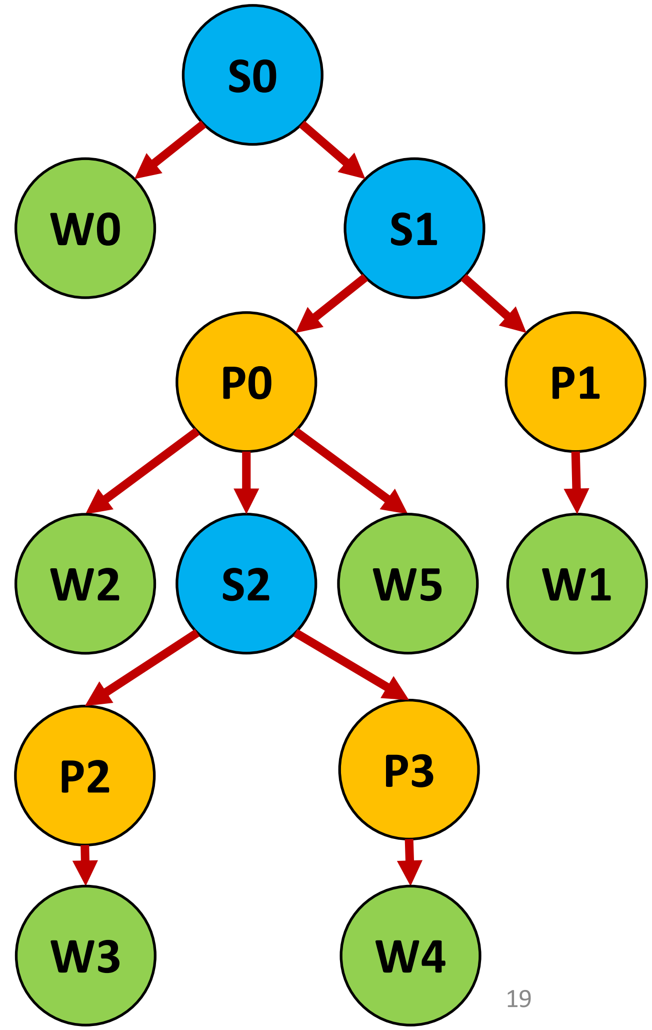
OSPG Construction

```
void main(){
  int* arr = init(&n);
#pragma omp parallel
  #pragma omp single
    mergeSort(arr, 0, n);
}
```



OSPG Construction

```
void mergeSort(int* arr, int s, int e){
  if (n <= CUT_OFF)
     serialSort(arr, s, e);
  int mid = s + (e-s)/2;
  #pragma omp task
     mergeSort(arr, s, mid);
  #pragma omp task
     mergeSort(arr, mid+1, e);
  #pragma omp taskwait
  merge(arr, s, e);
}
```



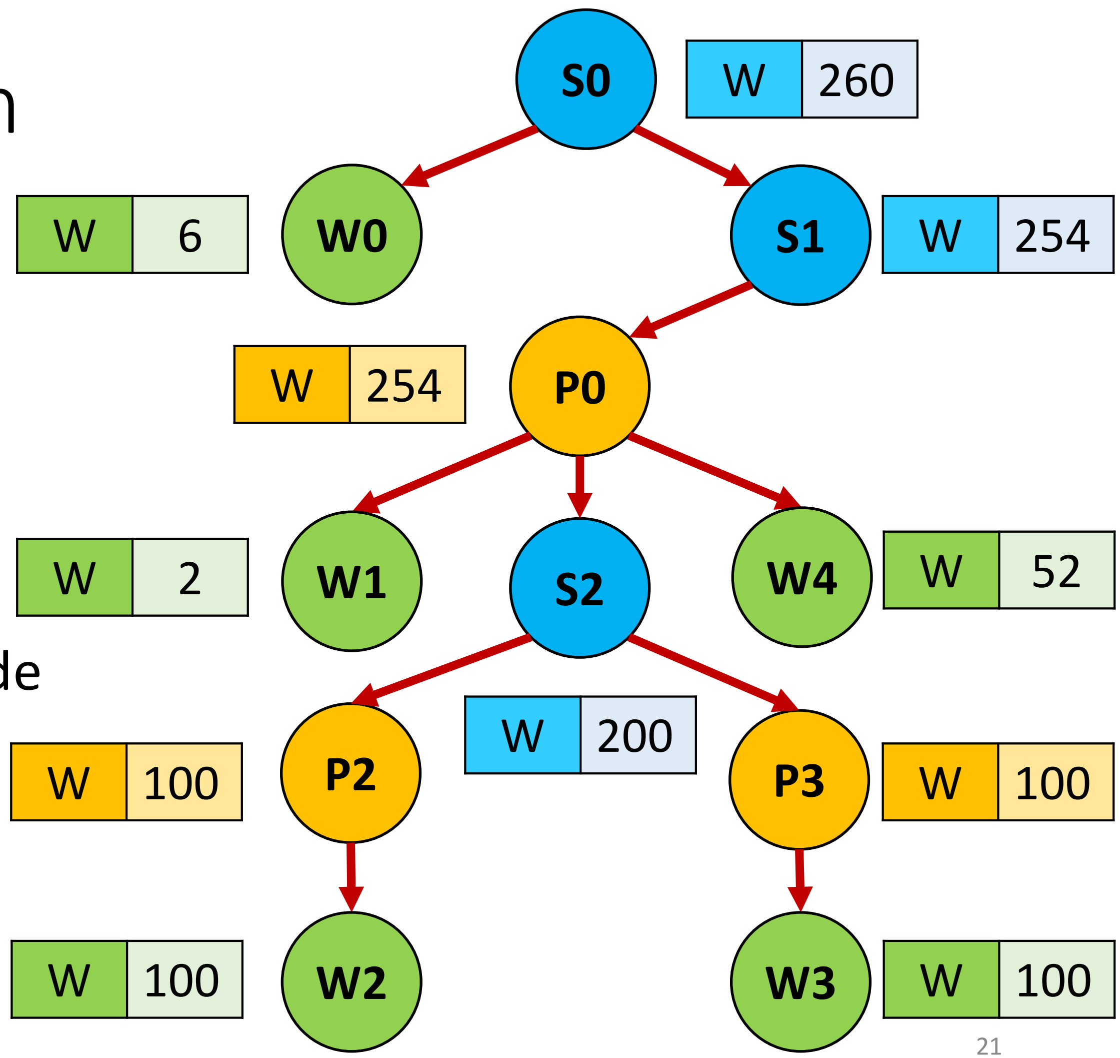
RUTGERS

Parallelism Computation Using OSPG

Compute Parallelism

Measure work in each Work node with fine grained measurements

Compute work for each internal node



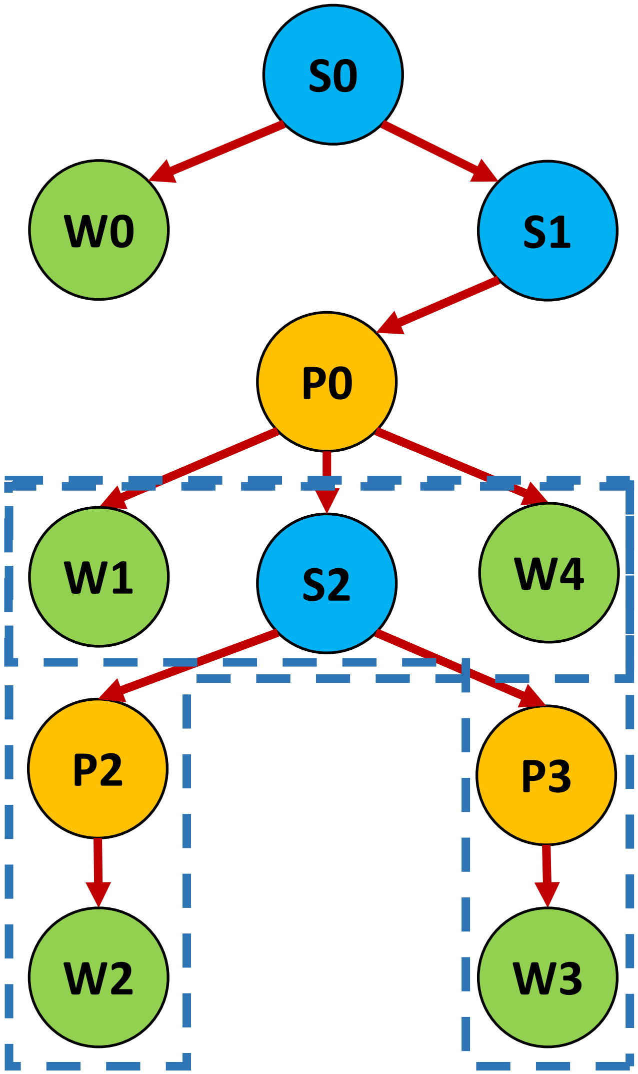
RUTGERS

Compute Serial Work

Measure work in each Work node

Compute work for each internal node

Identify serial work on critical path



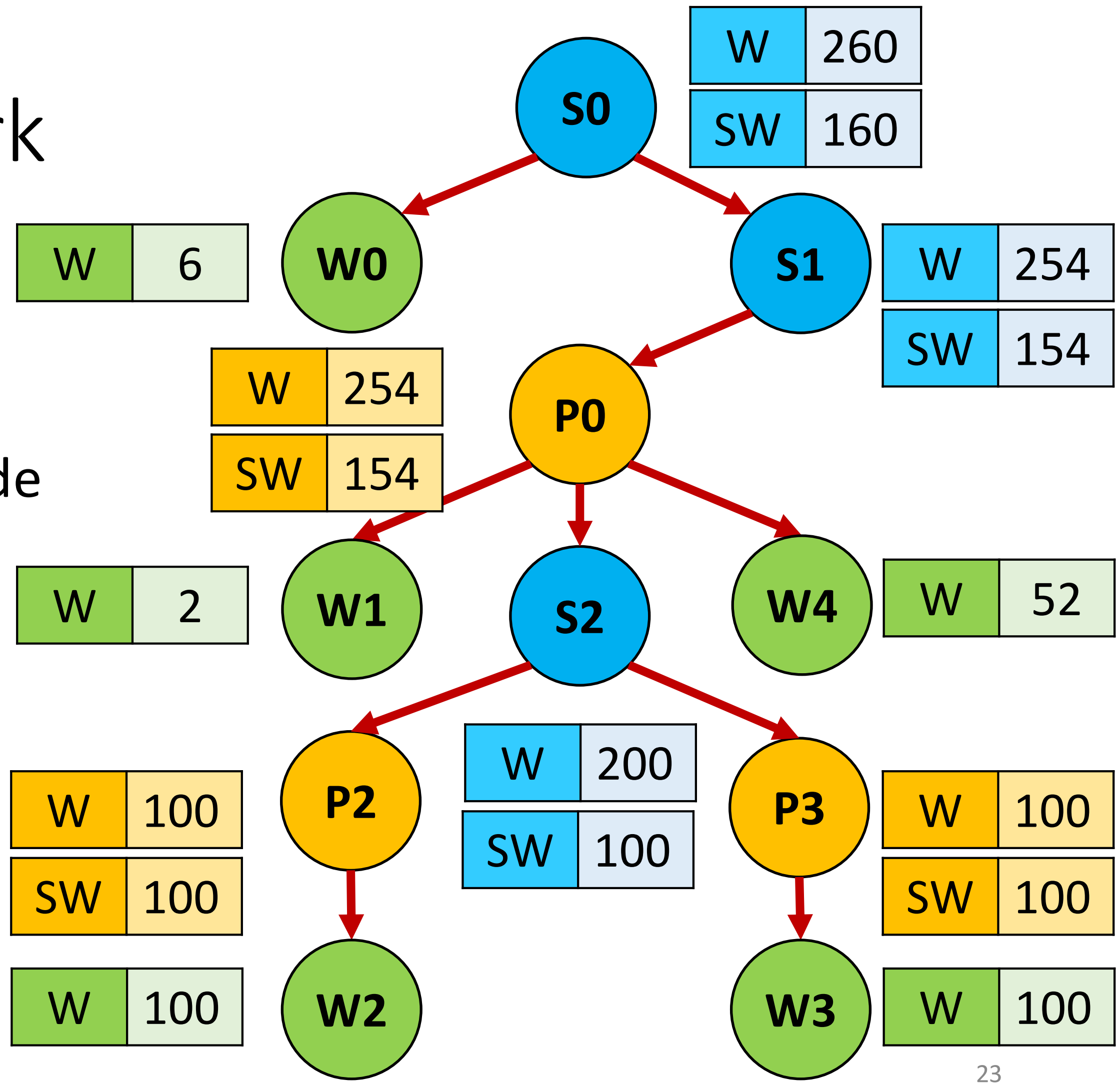
RUTGERS

Compute Serial Work

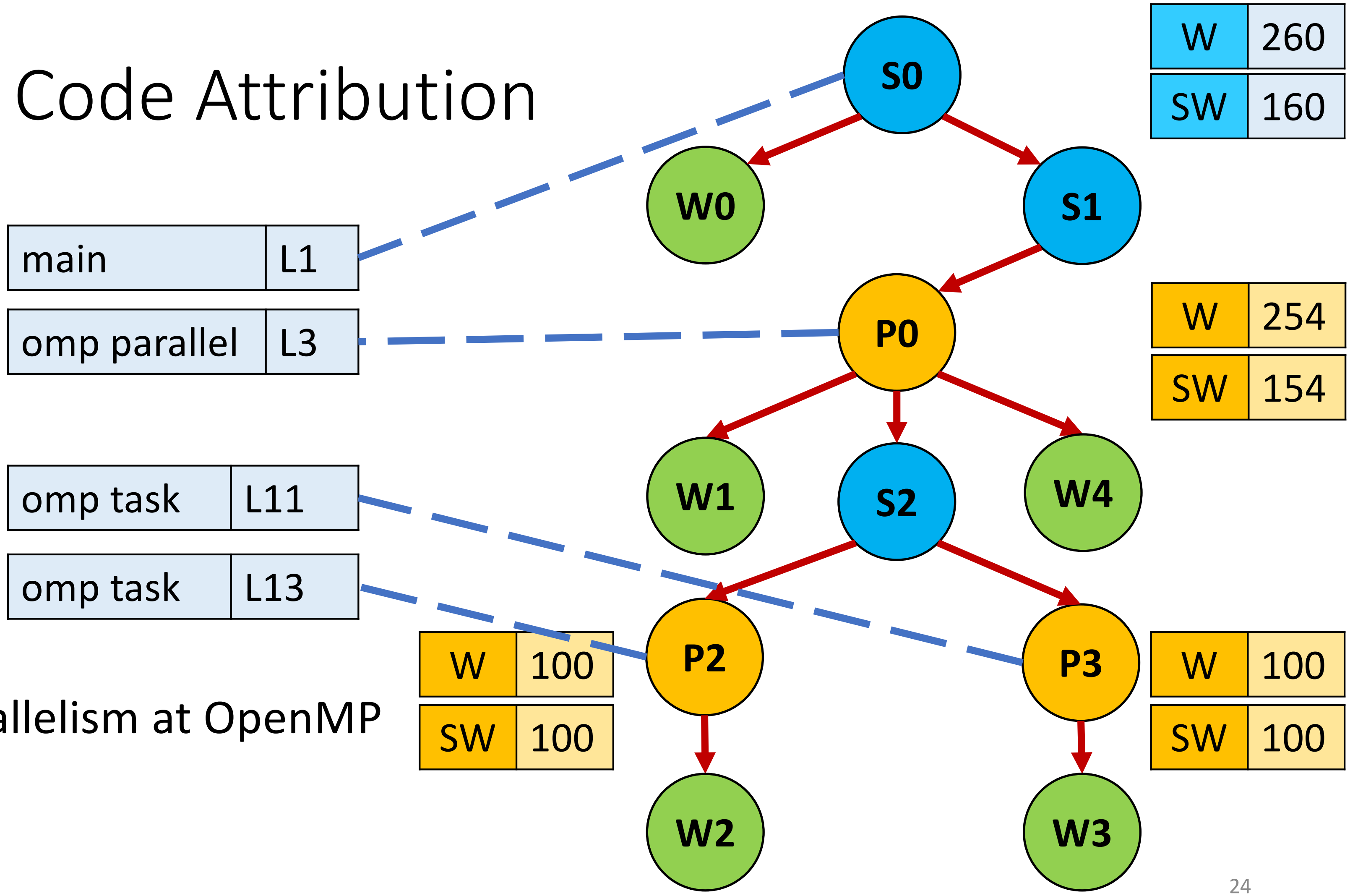
Measure work in each Work node

Compute work for each internal node

Compute serial work for each Internal node



Source Code Attribution

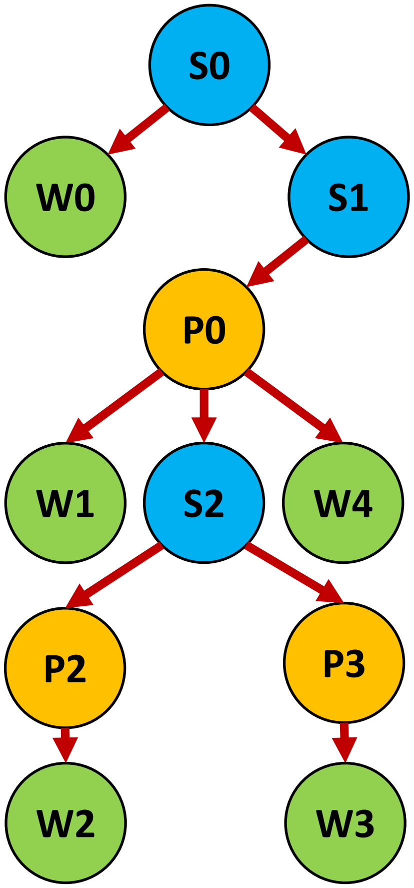


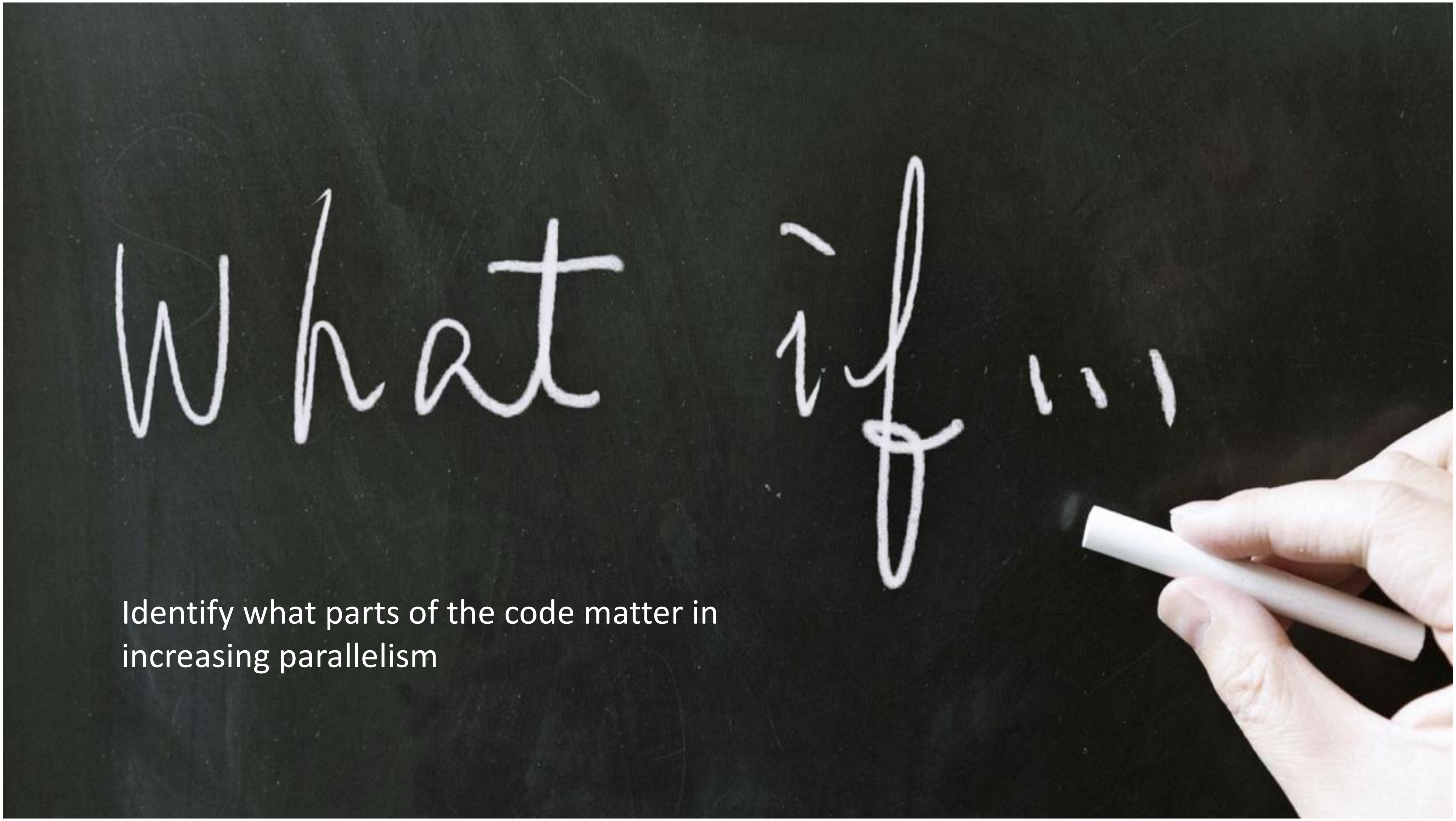
Aggregate parallelism at OpenMP constructs

RUTGERS

Parallelism Profile

Line Number	Work	Serial Work	Parallelism	Critical Path Work %
program:1	260	160	1.625	3.75
omp parallel:3	254	154	1.65	33.75
omp task:11	100	100	1.00	62.5
omp task:13	100	100	1.00	0





Identify what parts of the code matter in increasing parallelism

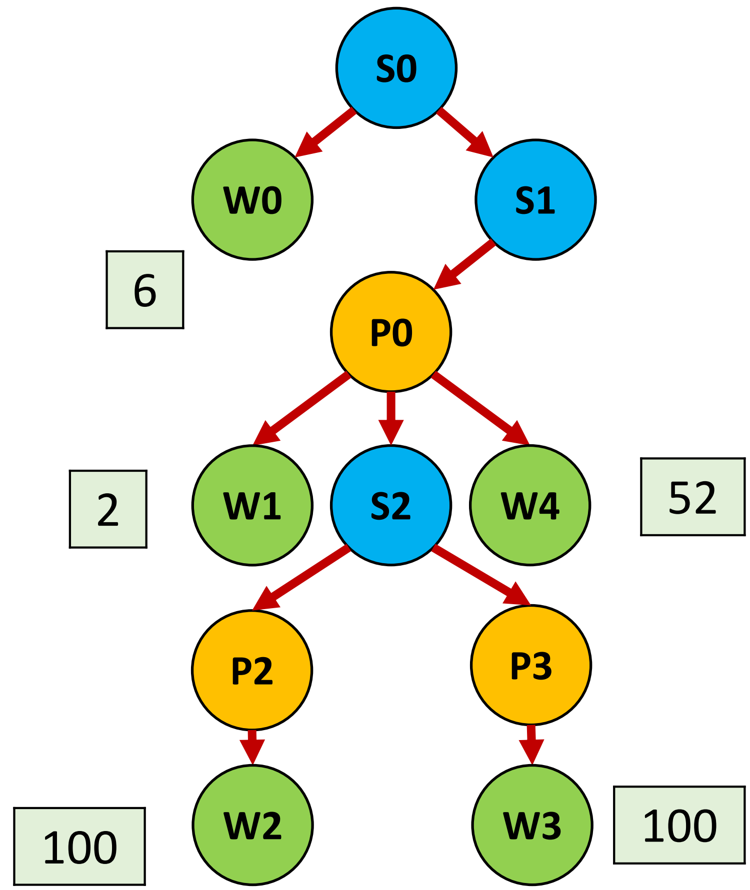
Adviser mode with What-If Analyses

Identify code regions that must be optimized to increase parallelism

Which region to select?

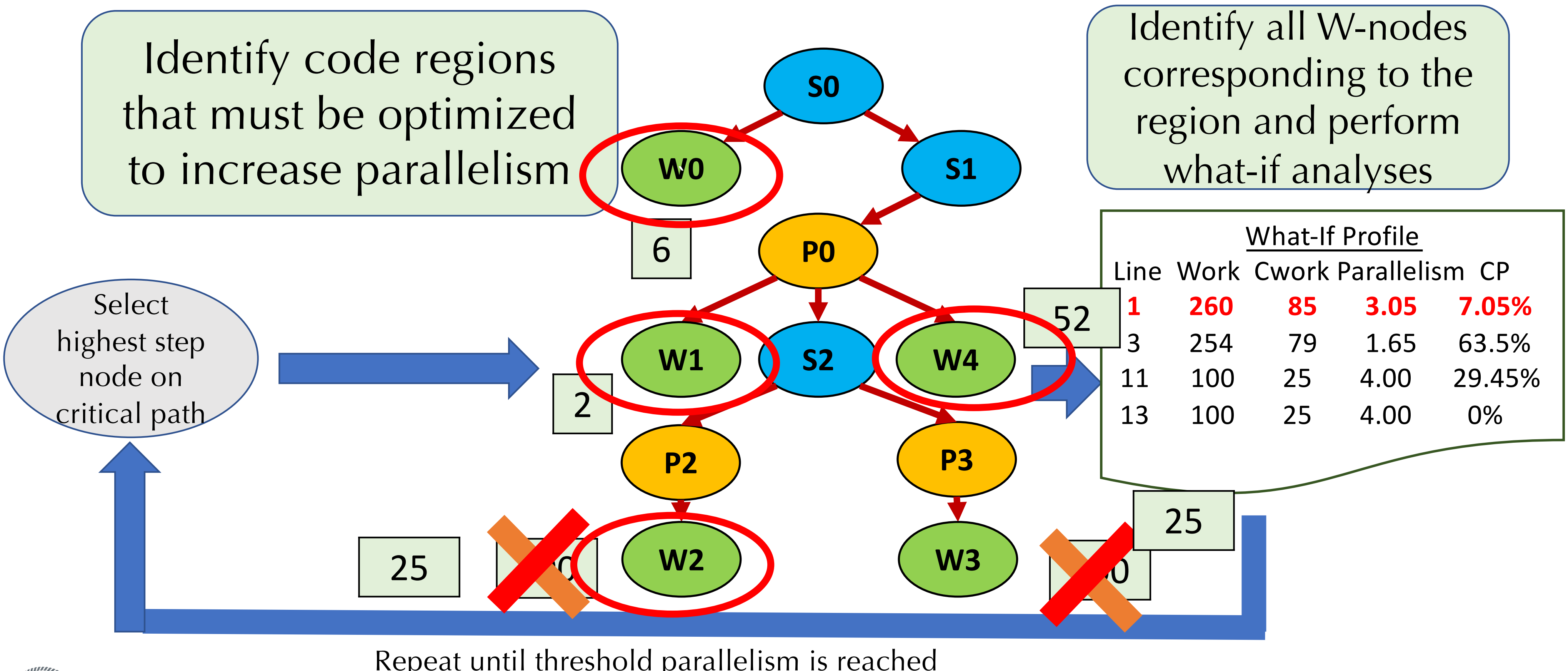
Select a region to optimize

Select step node performing highest work on critical path



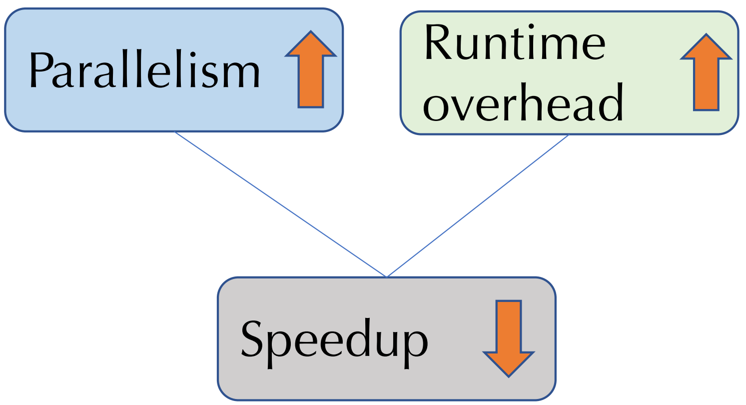
RUTGERS

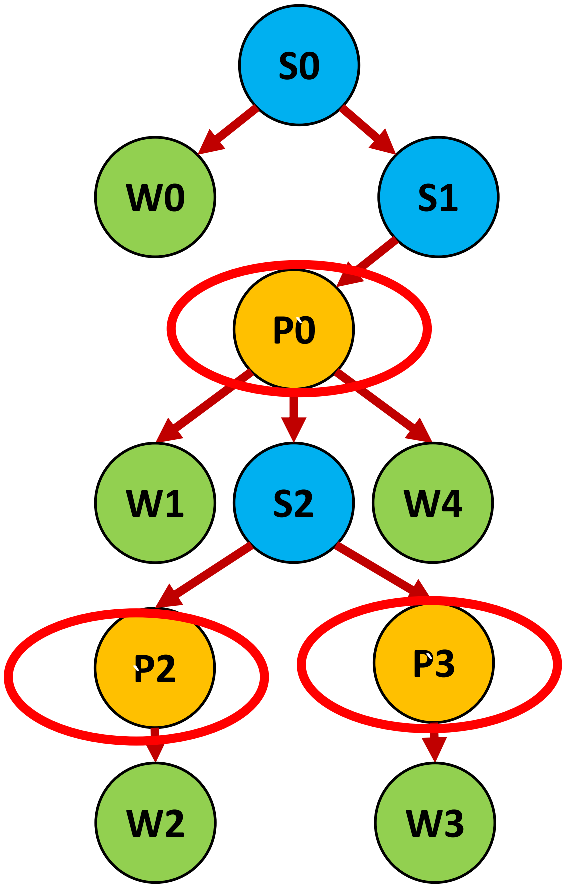
Adviser mode with What-If Analyses



RUTGERS

Tasking and Scheduling Overhead

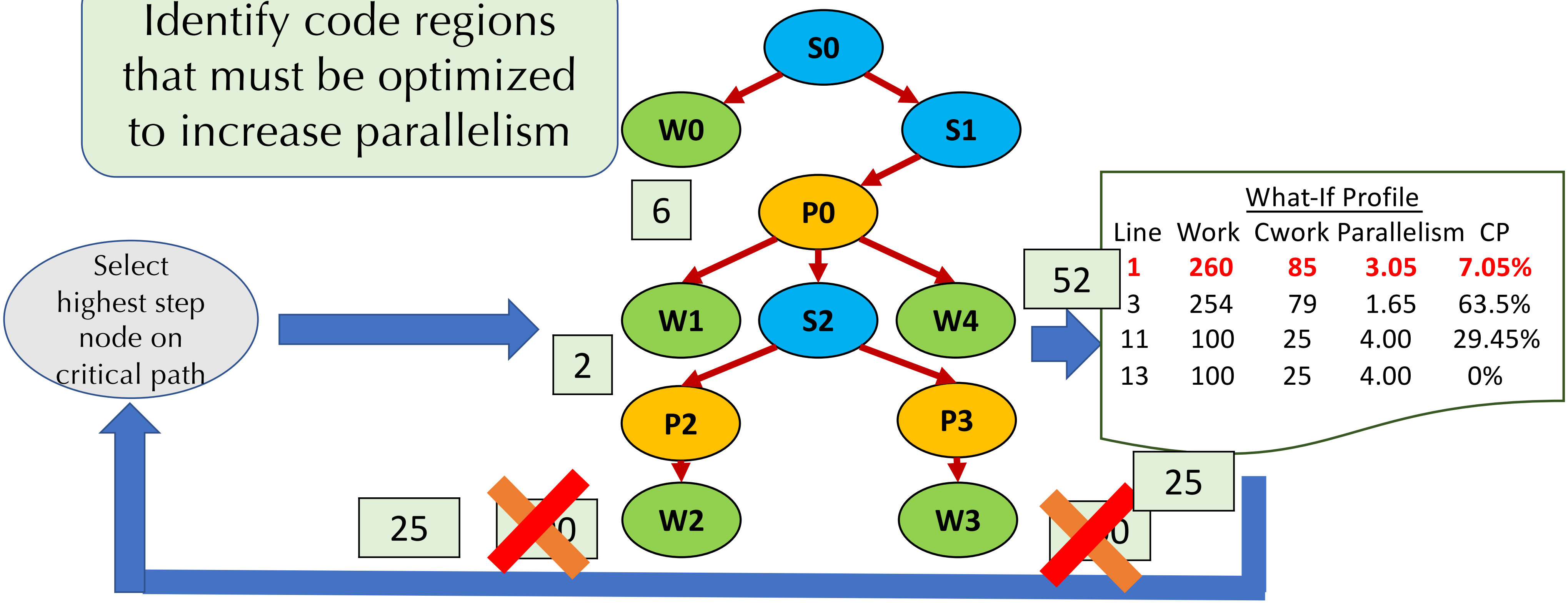




RUTGERS

Adviser mode with What-If Analyses

Identify code regions that must be optimized to increase parallelism



What-If Profile

Line	Work	Cwork	Parallelism	CP
1	**260**	**85**	**3.05**	**7.05%**
3	254	79	1.65	63.5%
11	100	25	4.00	29.45%
13	100	25	4.00	0%

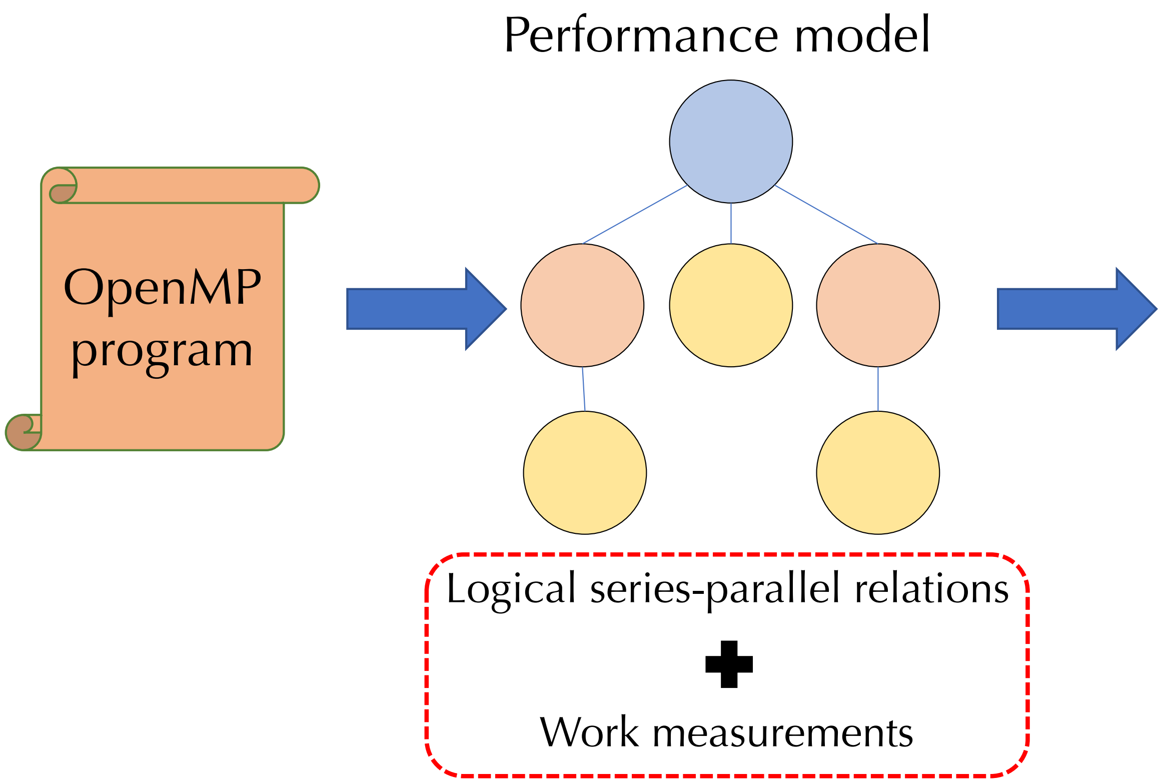
Repeat until threshold parallelism is reached

OR

Work of highest step node < K * average tasking overhead

RUTGERS

Recap



Parallelism Profile

Line	Work	Cwork	Parallelism
12	160	130	1.23
….	….	….	…..

What-if Regions

Region	Parallelization
12 - 14	4X
…. ….	…. …..

What-if Profile

Line	Work	Cwork	Parallelism
12	160	130	16.12
….	….	….	…..

Differential Analysis to Identify Secondary Effects

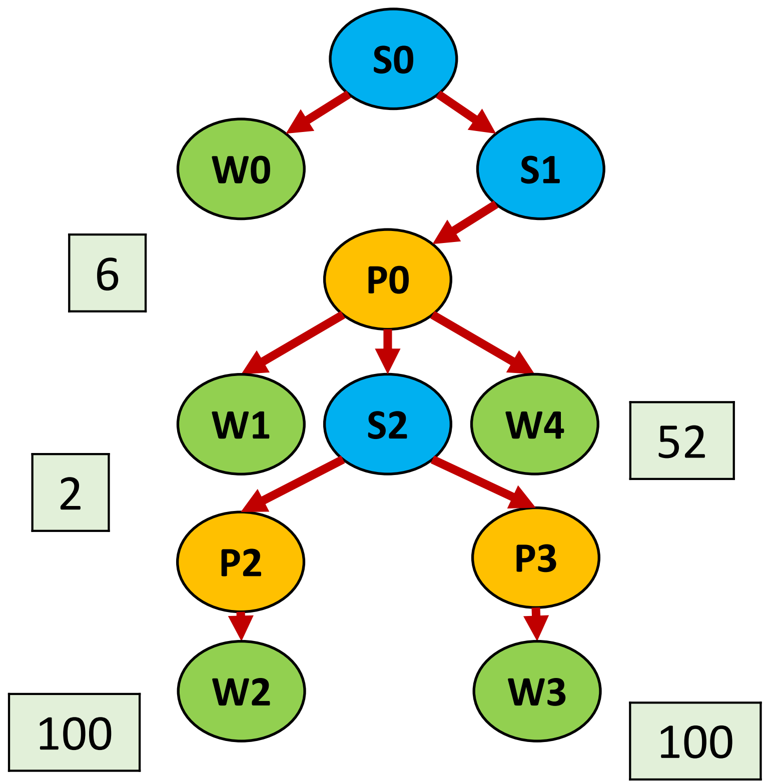
Beyond Parallelism - Secondary Effects

- Program can have high parallelism, but low speedup
  - Secondary effects of parallel execution on hardware

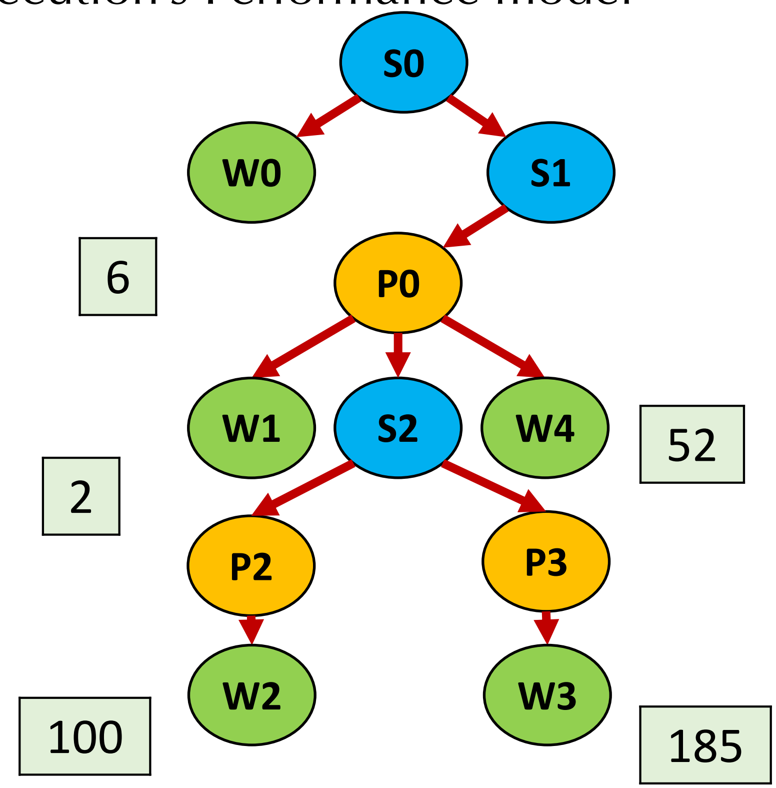
- Contention for a system resource
  - Cache – False sharing
  - Memory – High remote memory accesses
  - LLC misses - Reduced locality
  - Processor to data affinity

Differential Analysis

Oracle Performance model



Parallel Execution's Performance model



Work inflation in region with secondary effects

RUTGERS

Inflation over Multiple Metrics

Differential Counters

Cycles
HITM
Remote DRAM
accesses
……

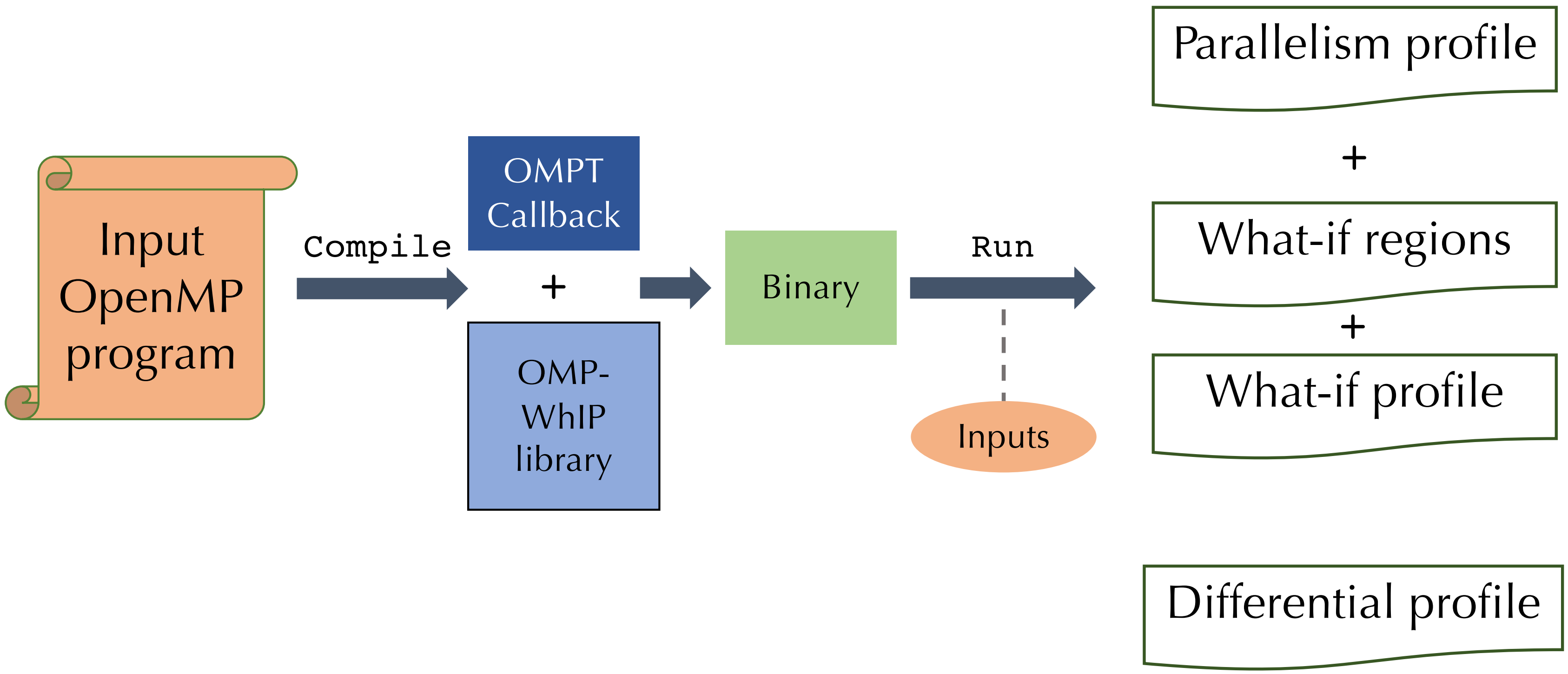
Differential Profile

Regions	Cycles	HITM	RemDRAM
main	**4.19X**	**13.2X**	**1.1X**
2-4	**5.34X**	**17.8X**	1X
14-15	1.02X	1.03X	1X
15-16	1.03X	1.1X	1.01X

RUTGERS

Prototypes for OpenMP and Task Parallelism

OMP-WHIP for OpenMP programs: https://github.com/rutgers-apl/omp-whip/
TaskProf for Intel TBB programs: https://github.com/rutgers-apl/TaskProf2



Optimizing MILCmk

Initial Parallelism Profile

Parallelism Profile

File:Line	Parallelism	Cpath
main	**44.21**	**28.3**
vmeq.c:23	30.29	23.3
veq.c:28	32.83	19.55
vpeq.c:28	33.55	9.35
….	….	…..

What-if Profile

What-if Regions
funcs.c:81 – 91
funcs.c:60 – 67
funcs.c:47 - 54

What-if Profile

File:Line	Parallelism	Cpath
main	**89.89**	**21.3**
vmeq.c:23	30.29	25.2
veq.c:28	32.83	21.5
vpeq.c:28	33.55	11.5
….	….	…..

Optimizing MILCmk

```
QLA_Real
sum_V(QLA_ColorVector *d, int n)
{
  QLA_Real t=0, *r=(QLA_Real *)d;
  int nn = n*sizeof(QLA_ColorVector)/sizeof(QLA_Real);
  for(int i=0; i<nn; i++) t += r[i];
  return t/nn;
}
```

Replaced serial for loop with parallel_reduce

```
QLA_Real
sum_H(QLA_HalfFermion *d, int n)
{
  QLA_Real t=0, *r=(QLA_Real *)d;
  int nn = n*sizeof(QLA_HalfFermion)/sizeof(QLA_Real);
  for(int i=0; i<nn; i++) t += r[i];
  return t/nn;
}
```

```
QLA_Real
sum_D(QLA_DiracFermion *d, int n)
{
  QLA_Real t=0, *r=(QLA_Real *)d;
  int nn = n*sizeof(QLA_DiracFermion)/sizeof(QLA_Real);
  for(int i=0; i<nn; i++) t += r[i];
  return t/nn;
}
```

RUTGERS

Optimizing MILCmk

Initial Differential Profile

	Differential Profile		
File:Line	Cycles	rem HITM	rem DRAM
main	**3.0X**	**100.4X**	**84.8X**
veq.c:28-35	3.8X	55X	78X
vmeq.c:20-22	3.7X	102X	61X
vpeq.c:20-27	3.6X	91X	68X
….	….	…..	…..

- Inflation in cycles and remote DRAM accesses in 5 parallel_for regions
- parallel_for loops were repeated multiple times
  - Lack of affinity
- Optimized by replacing default partitioner with affinity partitioner

Increased the speedup of MILCmk from 2.2X to 6X

RUTGERS

Is it Useful?

We found it to be effective with numerous applications.

Currently in talks for tech transfer with the Intel Vtune team.

Open Source at
https://github.com/rutgers-apl/TaskProf2
https://github.com/rutgers-apl/omp-whip/

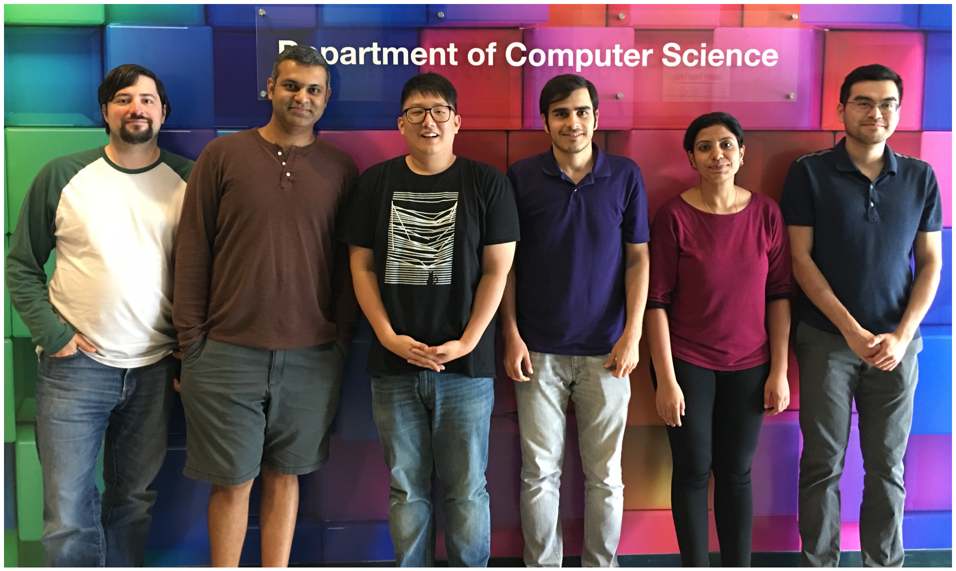
RUTGERS

Conclusion

- Make a case for measuring logical parallelism
- Series-parallel relations + fine-grained measurements ➔ a useful performance model for identifying scalability bottlenecks
- What-if analyses can help you identify regions that matter
- Differential analyses to identify regions having secondary effects
- Applicable to wide variety of programming models with appropriate series-parallel graphs

RUTGERS

Develop Abstractions for Performance & Correctness



Alive-NJ: https://github.com/rutgers-apl/alive-nj/

TaskProf2: https://github.com/rutgers-apl/TaskProf2

OMP-WHIP: https://github.com/rutgers-apl/omp-whip/

CASM-Verify: https://github.com/rutgers-apl/CASM-Verify/

Other software prototypes from the
Rutgers Architecture & Programming Languages Group:
https://github.com/rutgers-apl/

RUTGERS